Hayes, Minnie and Tyugu

AI and Machine Learning

Oxford

MACHINE
INTELLIGENCE 12

MACHINE INTELLIGENCE

Machine Intelligence 1 (1967) (eds N. Collins and D. Michie) Oliver & Boyd, Edinburgh

Machine Intelligence 2 (1968) (eds E. Dale and D. Michie) Oliver & Boyd, Edinburgh

(1 *and* 2 *published as one volume in* 1971 *by Edinburgh University Press*) (eds N. Collins, E. Dale, and D. Michie)

Machine Intelligence 3 (1968) (ed. D. Michie) Edinburgh University Press, Edinburgh

Machine Intelligence 4 (1969) (eds B. Meltzer and D. Michie) Edinburgh University Press, Edinburgh

Machine Intelligence 5 (1970) (eds B. Meltzer and D. Michie) Edinburgh University Press, Edinburgh

Machine Intelligence 6 (1971) (eds B. Meltzer and D. Michie) Edinburgh University Press, Edinburgh

Machine Intelligence 7 (1972) (eds B. Meltzer and D. Michie) Edinburgh University Press, Edinburgh

Machine Intelligence 8 (1977) (eds E. W. Elcock and D. Michie) Ellis Horwood, Chichester/Halsted, New York

Machine Intelligence 9 (1979) (eds J. E. Hayes, D. Michie, and L. Mikulich) Ellis Horwood, Chichester/Halsted, New York

Machine Intelligence 10 (1982) (eds J. E. Hayes, D. Michie, and Y.-H. Pao) Ellis Horwood, Chichester/Halsted, New York

Machine Intelligence 11 (1988) (eds J. E. Hayes, D. Michie, and J. Richards) Oxford University Press, Oxford

Machine Intelligence 12 (1991) (eds J. E. Hayes, D. Michie, and E. Tyugu) Oxford University Press, Oxford

MACHINE INTELLIGENCE 12

Towards an automated logic of human thought

---

edited by

J. E. HAYES

Research Associate, Turing Institute

D. MICHIE

Chief Scientist, Turing Institute

and

Ė. TYUGU

Head of Software,
Institute of Cybernetics,
Estonian Academy of Sciences

CLARENDON PRESS · OXFORD
1991

Oxford University Press, Walton Street, Oxford OX2 6DP
Oxford New York Toronto
Delhi Bombay Calcutta Madras Karachi
Petaling Jaya Singapore Hong Kong Tokyo
Nairobi Dar es Salaam Cape Town
Melbourne Auckland
and associated companies in
Berlin Ibadan

Oxford is a trade mark of Oxford University Press

Published in the United States
by Oxford University Press, New York

© J. E. Hayes, D. Michie, and Ė. Tyugu, 1991

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of Oxford University Press

British Library Cataloguing in Publication Data
Machine intelligence.
12, Towards an automated logic of human thought
1. Artificial intelligence
I. Hayes, J. E. (Jean Elizabeth) 1928– II. Michie, Donald 1923–
III. Tyugu, Ė. (Ėnn) 1935–
006.3

ISBN 0-19-853823-5

Library of Congress Cataloging in Publication Data
Machine intelligence 12: towards an automated logic of human thought
/edited by J. E. Hayes, D. Michie, and Ė. Tyugu.
p. cm.
Includes index.
1. Artificial intelligence—Data processing. 2. Machine learning.
I. Hayes, Jean E. II. Michie, Donald. III. Tyugu, Ė. Kh. (Ėnn
Kharal'dovich), 1935– . IV. Title: Machine intelligence twelve.
Q336.M33 1991 006.3—dc20 90-7320

ISBN 0-19-853823-5

Typeset by Cotswold Typesetting Ltd, Cheltenham
Printed in Great Britain by St. Edmundsbury Press, Bury St. Edmunds

FOREWORD

Evgeni Velikhov
Vice President of the Academy of Sciences of the USSR

It is a pleasure to contribute an introduction to this twelfth volume of the international *Machine Intelligence* series. My own work has, at times, cast me in the scientific roles of experimenter, instrumentation designer, and administrator. In all these roles I have seen the growing pervasiveness of the new tools of information technology. As a visitor to the 1987 meeting in Milan of the International Joint Conference on Artificial Intelligence I received a vivid impression of the role that machine intelligence in particular seems destined to play in this, the final decade of this century. Economic growth is increasingly dependent on new technologies in which the intelligence of machines plays a leading role. The concept of machine intelligence itself acquires a new semantic content. This is demonstrated by the evolution of new disciplines such as mechatronics as well as by the increasing importance of intelligent tools in manufacturing. It seems extremely important for the future of the human species that the mind that machines develop grows faster than the muscles, that is the energy parameters.

Looking at the eleven previous impressive volumes of *Machine Intelligence* one observes that the MI conferences have covered a significant part of the world including the Soviet Union where, before MI-12, the ninth MI conference was also held in 1977. I believe that the style and the content of the *Machine Intelligence* series will continue to reflect the much needed dialogue between various societies.

PREFACE

For almost twenty-five years I have, as editor of these volumes, presided over the inquisitiveness of the newly arrived. The young delight to get their noses into everything. But unbounded promise must sooner or later confront the emergence of what in ecology and in entrepreneurial commerce are known as 'niches'. Possibly, machine intelligence has, during all this time, been sleepwalking towards its own true niche. In any case, the series must now select one. Along what line do we see the future commitment of the *Machine Intelligence* series?

In 1986 a Steering Committee was formed to set a direction and to initiate the formation of an international editorial board with an executive editor and two associate editors to support the work and to organize the workshops themselves. These will resume their initial annual tempo. As editor-in-chief I am privileged to welcome our future executive editor Dr Stephen Muggleton. With regard to directions, the central theme will be the design of automated support for intellectual discovery and its application. Sophistication of computing aids is a conspicuous feature of today's scientific scene. From the astrophysicist's supercomputer to the field worker's pocket machine, the race has been to automate every function but one. That one is scientific reasoning itself, whilst AI has been the laggard.

More than a quarter of a century ago, the Nobel Prize-winning chemical microbiologist Joshua Lederberg had a vision of intelligent machines as partners in the scientific quest. In Stanford's DENDRAL project he initiated the first inroad into organized empirical enquiry. The tools of that time were too weak to accomplish more than the planting of a series of signposts, some of which appear in earlier MI volumes. Among these the MetaDENDRAL module set a crucial pointer to the need, reflected in this volume, to mechanize the inductive as well as the deductive component of the cycle of scientific inference.

A modern scientist can fairly be described as an inductive agent loaded to breaking point by complexity. Reporting from a sector where the strain is especially severe, Ross King describes in this volume an application of computer induction to the prediction of protein folding. Elsewhere he has written that 'it was once possible to discover the meaning of new data by carefully examining it by eye'. That time is, of course, long past. Today, decision supports from statistical data analysis are pressed into service. But now even these impressive constructions are proving inadequate to such complex requirements as those of biotechnology for empirical theories of structure–activity relationships,

and the requirement for better models of our planet as a basis of rational plans for the next century.

At some stage in the mechanized analysis of any sufficiently complex problem, further progress (as indicated for example in the chapter by Mozetič, Bratko, and Urbančič) has to await intelligible mechanization of the underlying relations of cause and effect. The wheel here comes full circle. John McCarthy's paper of just 30 years ago, 'Programs with common sense', placed at the core of AI's coming tasks the need for a machine-oriented logic capable of expressing causality in everyday life. Progress has subsequently been made, but in its unrestricted form McCarthy's plan remains ambitious. By restricting the aim of mechanizing causal reasoning to defined domains of scientific study we may find both a measure of tractability and also uncommon rewards.

Not the least reward must surely be the sense of mutual usefulness among disciplines, which forms the living cement of our invisible college.

June 1990

Donald Michie
Editor-in-Chief

ACKNOWLEDGEMENTS

Our grateful thanks are due to the Estonian Academy of Sciences, in particular the Institute of Cybernetics who hosted the meeting which provided the basis for this volume in their beautiful city of Tallinn. Thanks are also due to Knowledgelink (Intelligent Terminals Limited), to the Turing Institute for help with travel and resources for the work of editing the papers printed here, and the many who helped with the conference organization and the editorial process. In particular we would like to thank Irene Brebner, Pearl Guthrie, Liina Kesküla, and Catherine McCrae.

CONTENTS

MECHANICS OF KNOWLEDGE PROCESSING

1. Modularity of knowledge 3
   Ė. Tyugu
2. Propositional logic programming 17
   G. Mints
3. Computational models in Prolog 39
   A. A. Lomp
4. On the construction of unifying terms modulo a set of substitutions 49
   S. Lange
5. Plausible inference and negation in Horn clause logic 55
   T. B. Niblett
6. A note on first-order theories of individual concepts and propositions 79
   B. Arbab

INDUCTIVE FORMATION OF PROGRAMS AND DESCRIPTIONS

7. Inverting the resolution principle 93
   S. H. Muggleton
8. Non-monotonic learning 105
   M. Bain and S. H. Muggleton
9. Interactive induction 121
   W. Buntine and D. Stirling
10. Models of inductive syntactical synthesis 139
    J. Barzdin, A. Brazma, and E. Kinber

OPTIMALITY AND ERROR IN LEARNING SYSTEMS

11. Deriving the learning bias from rule properties 151
    J. G. Ganascia
12. Error tolerant learning systems 169
    C. Sammut
13. Use of sequential Bayes with class probability trees 187
    D. Michie and A. Al Attar

QUALITATIVE REPRESENTATIONS OF KNOWLEDGE

14. Exploring structures: an exercise in model-based interpretation and planning 205
    I. Bratko

15. Learning of causality by a robot 225
P. MOWFORTH and T. ZRIMEC
16. A qualitative way of solving the pole balancing problem 241
A. MAKAROVIC
17. Varying levels of abstraction in qualitative modelling 259
I. MOZETIČ, I. BRATKO, and T. URBANČIČ

APPLICATIONS AND MODELS OF KNOWLEDGE ACQUISITION

18. Information content of chess positions: implications for game-specific knowledge of chess players 283
J. NIEVERGELT
19. PROMIS: experiments in machine learning and protein folding 291
R. D. KING
20. Design of knowledge processing systems—principles and practice 311
S. OHSUGA

INDEX 331

MECHANICS OF KNOWLEDGE PROCESSING

1

Modularity of Knowledge

Ė. Tyugu
Institute of Cybernetics,
Estonian Academy of Sciences, USSR

When you lose a game of chess to a computer then don't pretend that you didn't think at the game.

S. Maslov

Abstract

Merging different kinds of knowledge in problem-solving is discussed. Several formal calculi are considered as knowledge representation means, and uniting calculi in the NUT programming system is described.

This paper has been inspired by the last book of Sergei Maslov [1] where he described a tower of deductive systems as a representation of scientific knowledge about the world. He illustrated the usage of formal calculi by numerous examples from biology, economics, and technology.

1. INTRODUCTION

We shall discuss here modularity of knowledge in the large. This is not breaking the whole of available knowledge into uniformly represented parts. We are interested in merging various kinds of knowledge and using them all together for achieving some hard goal. The question is: 'How to combine different knowledge representations and handling techniques in problem-solving systems?'

Experience shows that no universally efficient knowledge representation and handling technique exists. On the contrary—a number of very different methods have been developed for solving practically interesting problems in various domains. When considering human intelligence one can also distinguish basically different knowledge-handling mechanisms that are associated with the left and right parts of the brain, that is with logical and intuitive ways of thinking. We can hope that using different knowledge-handling methods in combination will help us to improve the intellectual capabilities of Artificial Intelligence (AI) systems designed for practical applications.

At first glance, blackboard systems seem to be a good example of modularity of knowledge in the large. This is true only when we are considering aspects of implementation. Blackboard systems provide a

framework for implementing modularity of knowledge, but they do not help us in finding suitable forms of knowledge representation.

This paper is based on an assumption that any *knowledge system* (KS) which is knowledge representation plus inference engine can be reduced to a formal calculus that adequately represents knowledge processing in this KS. This statement become trivial as soon as we loosen the requirement of adequacy: on a sufficiently low level we can use Turing machines or Post's systems for representing information processing in computers.

Secondly, this paper elaborates on an observation that any successful AI system contains more than one KS, that is it is based on several calculi combined with each other in non-trivial ways. The latter means that there is no obvious natural way to build a single calculus preserving the requirement of adequacy. Putting together calculi of various knowledge systems mechanically would give us a tower of Babel of formal languages—a calculus that is incomprehensible as well as inefficient.

Nevertheless, all purely procedural forms of knowledge can be represented by a single calculus. Any representation of computable functions together with application rules for functions can be used for this purpose (Markov's normal algorithms, recursive functions, etc.). Let us call the calculus chosen the calculus of computable functions (CCF). It seems that CCF is present in any sufficiently general knowledge-based system, because procedural knowledge is a convenient means for providing extensibility to a knowledge-based system.

As soon as we intend to apply procedural knowledge automatically, another calculus is needed for invoking programs. We have good examples of systems where two calculi are used, one for procedural knowledge and another for control of computations. PROLOG combines Horn clause logic with CCF, structural synthesis of programs uses intuitionistic propositional calculus (IPC) for control and CCF for the procedural part.

We have developed programs that contain more than two KS's. The system PRIZ [2] and MicroPRIZ [3], besides the CCF and IPC, also use a rewriting system as a user-friendly front end. It transforms specifications written in a high-level specification language into a set of specific axioms of a formal calculus.

One more calculus is added to those mentioned above in the systems ExpertPRIZ and NUT [4]. ExpertPRIZ is an extention of MicroPRIZ that combines inductively built knowledge bases supporting simple decision-tree logic with the three calculi of PRIZ. The NUT system combines first order calculus of productions with PRIZ calculi.

2. FORMAL CALCULI AS KNOWLEDGE REPRESENTATION MEANS

There are many papers on using logic for knowledge representation [5].

Our thesis is that in all cases when we use knowledge, making inferences step by step, we can build a calculus that represents this knowledge and the inference engine. It is obvious that this thesis cannot be proved formally. However, looking at numerous examples we can find good evidence in favour of this thesis. First of all, making inferences means using knowledge in a deductive way, and in his book [1], Maslov has described a number of calculi, called also deductive systems, which are formalizations of knowledge in various domains. He has defined the concept of calculus in a very general way that suits us well:

> There are a certain number of initial objects and a certain number of rules for generating new objects from the initial objects and from those already constructed. To put it another way: There are an initial position (state) and 'rules of the game' (rules for transition from one state into another). A system of this kind is called a *deductive system*, or a *calculus*.

Let us consider semantic networks as an example of knowledge systems and let us try to build a calculus for them. There are various kinds of semantic networks and different inference mechanisms for working on these networks.

Bearing in mind that any semantic network is a marked graph, we can represent it as a collection of arcs. For instance, Figure 1 shows a representation of explicit and implicit time-relations in the following text:

> John must pick up his report in the morning and have a meeting after lunch. After the meeting he will give the report to me.

The arcs of the network will be objects of the calculus we are building. In this example we have the following objects:

before (lunch, morning)
after (morning, lunch)
after (lunch, have a meeting)
after (have a meeting, give)
at-the-time (morning, pick up)

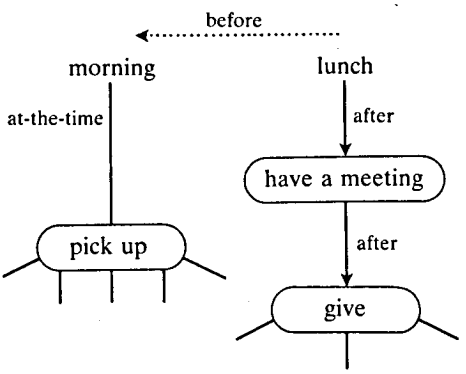


Figure 1. Time-relations.

Sometimes it is more convenient to use another representation of the network, looking at nodes as relations that bind all their neighbouring nodes. In this case the objects will be nodes, used as relations between the other nodes (their neighbours).

Inference on a semantic network is done by propagating facts (or, more generally, 'pieces of knowledge') along the arcs of the network. As a result, the network itself or the marking of its nodes is changed. Inference rules, as usual, are schemes of the following form:

$$S_1, \ldots, S_k \vdash S$$

where objects $S_1, \ldots, S_k$ are premises and the object S is a conclusion. In our case there are rules for transitivity of some time-relations, for instance:

before (x,y), before $(y,z) \vdash$ before (x,z).
after $(x,y) \vdash$ before (y,x).
at-the-time (x,z), before $(y,z) \vdash$ before (y,x).

Applying these rules we can make inferences like:

after (lunch, have a meeting) ⊢ before (have a meeting, lunch) at-the-time (pick up, morning), before (lunch, morning) ⊢ before (lunch, pick up)

and add new arcs to the graph.

3. HOW TO COUPLE CALCULI?

First of all, let us consider briefly the implementational aspects of coupling different calculi. There are well-known ways of implementing a system which consists of several interacting experts:

(1) building a network of communicating actors (processes, experts) [6];
(2) using blackboard architecture [7, 8, 9];
(3) using broadcasting as a means of communication between the experts.

All these ways can be used for writing knowledge systems, each of which will be then represented as an 'expert' with its own knowledge representation forms and inference engine.

A network of experts can be efficient for loosely coupling several knowledge systems. Object oriented programming is suitable for this purpose, because message passing can be directly used for communication between the KS's. Actually, any tools for programming communicating sequential processes can be used.

Another way to achieve the same goal is to use 'broadcasting'. In this way we can organize knowledge systems to show collective behaviour that mimics the behaviour of a group of co-operating human experts.

The closest co-operation between the knowledge systems can be provided in a blackboard system. In this case, a considerable amount of knowledge (the blackboard) is visible for all knowledge systems. The question remains, how does each KS understand the knowledge on the blackboard? But this is one of the principal questions that need to be considered when writing the KS.

In order to choose one or other of the architectures we must consider the principles of writing a KS. Some useful hints can be obtained from pure logic.

In proof theory we can find examples of successful decomposition of theories. Roughly speaking, sometimes a theory can be split into several parts, so that different inference methods can be applied and efficiency of search can be significantly improved. The following two techniques are worthy of mention:

(1) constructing a set of admissible inference rules;

(2) using a metatheory.

Both these techniques have analogies in knowledge-based systems.

Yet another useful way of combining calculi comes from logic. Let us take a constructive non-categoric theory (that is, a theory that has more than one model). Models of constructive theories can again be considered as calculi. So we have a non-trivial relation of interpretation ('to be a model of') between the calculi. Probably the relation of interpretation is the most widely used relation between the calculi in knowledge-based systems.

In papers on algebraic data types which are represented as heterogeneous algebras, we can find a number of relations between algebras [10]: abstraction, concretization, extension, restriction, enrichment, etc. To an extent, these relations are also meaningful for calculi of knowledge systems.

4. UNITING LOGIC WITH PROCEDURAL KNOWLEDGE

A good example of uniting logic and procedural knowledge is PROLOG. It combines Horn clause logic (HCL) with a procedural knowledge system (CCF). Connection between the HCL and the CCF in PROLOG is established through the realization of functional constants and some predicates as pre-programmed procedures.

A text in PROLOG, that is, the logical part of a PROLOG 'program' consists of clauses

$$A \& \ldots \& B \rightarrow C$$

where every negative atom $A, \ldots, B$ must have a predicate that also occurs in a positive position of some clause or that has a clearly procedural realization. Let us consider an atom $P(t_1, \ldots, t_n)$. The terms $t_1, \ldots, t_n$ are either variables, or they are expressions containing constants that must also be realized in the CCF. The elegance of PROLOG comes from the successful combination of HCL and CCF. From one side, the logical part (HCL) uses the procedural part (CCF) only as an oracle and exercises a complete control over the order of execution of procedures. From the other side, every predicate occurring in the positive position of a clause can be used in the same way as pre-programmed predicates. That means that new 'realizations' of predicates can be written in PROLOG, and only procedurally 'tricky' programs must be implemented directly in CCF. These are the programs that, for instance, interact with the computational environment of a PROLOG system or use terms $t_1, \ldots, t_2$ of a predicate in a way that is out of the scope of PROLOG logic (for example as higher order objects).

We can observe another case of uniting logic with CCF in the PRIZ system [2]. In this case, the logic is intuitionistic propositional calculus (IPC). For readers unfamiliar with the IPC I should like to stress that it is more complex than classical propositional calculus—roughly speaking, because its formulas can more seldom be reduced to each other than in classical logic. (The equivalence $\neg\neg a \leftrightarrow a$ is not valid in IPC.) The proof-searching in IPC is a P-space complete problem. One can say that formulas of IPC represent types of complicated objects like functions of functions of functions and so on.

Actually, only a fragment of IPC is used as a logical language in the PRIZ system. This fragment contains the formulas of the following form:

$$X_1 \& \ldots \& X_k \rightarrow Y,$$

or, abbreviated

$$X \rightarrow Y; \tag{1}$$

and $(U^1 \rightarrow V^1) \& \ldots \& (U^m \rightarrow V^m) \rightarrow (X \rightarrow Y)$,

or, abbreviated

$$(U \rightarrow V) \rightarrow (X \rightarrow Y) \tag{2}$$

This fragment has the expressive power of the complete IPC in the sense that, for the purposes of theorem-proving, any formula of IPC can be represented as a set of formulas of the form (1), (2) by introducing new variables.

Knowledge about a concept (object, situation, or process) is represented in the PRIZ system as a set of formulas of the form (1), (2) and it is called a computational model. This knowledge is used for

solving problems, employing the schema

computational model → proof → program,

where proof of solvability of a problem is derived from a computational model in IPC. This constructive proof is used as a schema of a program that solves the problem.

Connection between the IPC and the CCF is established by giving a realization of any derivable formula of IPC in the form of a formula of CCF. (This realization is a computable function.) If we write the function associated with an implication under the arrow of the implication, then we shall have the formulas that show explicitly the connection between the logic and CCF in PRIZ:

$$X \xrightarrow[f]{} Y,$$

$$(U \xrightarrow[g]{} V) \to (X \xrightarrow[F(g)]{} Y).$$

Functions associated with the formulas of a computational model (which are axioms) must be predefined. Functions for derived formulas are constructed at every step of derivation from the functions associated with the premises.

As in PROLOG, the logical part of the PRIZ system (IPC) completely controls the order of execution of programs represented in CCF. Besides that, the logical part shows precisely how types of objects of CCF are related to each other, and these types can be of any finite order.

5. CALCULUS OF INHERITANCE

While PROLOG is still more or less suitable for direct representation of knowledge by users, the logical calculus of PRIZ is completely inappropriate for this purpose. In order to facilitate knowledge representation by human experts, a metalanguage for specifying theories represented in IPC was developed and it is used as an input language of the PRIZ system. This is an example of a high-level knowledge representation language with the semantics precisely described by a simple calculus, see also [11].

A specification in this language is of the form

$$a : t$$

where a is a new identifier and t is a type specifier. It can be either a direct specifier or an inheritance specifier.

The *direct specifier* has the following form:

$$(copy\ c_1; \ldots; copy\ c_m; x_1 : t_1; \ldots; x_k : t_k; \langle \text{relations} \rangle).$$

A new object specified by the direct specifier

(1) inherits all properties of objects $c_1, \ldots, c_m$;

(2) gets components $x_1, \ldots, x_m$ with the properties specified by $t_1, \ldots, t_k$ (any $x_i : t_i$ is a specification);

(3) obtains properties given by relations.

The relations are either equations or axioms of IPC with given realizations. Outside the object a, its components $x_1, \ldots, x_k$ will have compound names $a.x_1, \ldots, a.x_k$.

The *inheritance specifier* has the form:

$$y\, y_1 = v_1, \ldots, y_n = v_n$$

where y is the name of an object which has already been specified, $y_1, \ldots, y_n$ are some of the components of the object y, and $v_1, \ldots, v_n$ are either constants or names of some other objects.

There are predefined objects *numeric*, *bool*, and so on which can be used as primitive specifiers. One of them, *any*, is of special importance. It enables us to build generic objects. A component y_i of this type can be used in an equation $y_i = v_i$ of the inheritance specifier with the object v_i of any other type, and then it acquires for itself the type of v_i. For instance, having specified

stack: (. . . ; elem: *any*; . . .)

and provided a pointer is an object, we can build a stack of pointers

pointerstack: stack elem = pointer.

This language enables us

(1) to build hierarchically structured objects:

(2) to use multiple inheritance;

(3) to describe specific features of objects by means of relations.

This last feature is still obscure. The explanation is given below.

Let us define a calculus in which, for any specification S, a computational model $M(S)$ (that is, a set of axioms of the form (1) (2)) is derivable which determines the possible computations specified by S.

First of all, for any object x we introduce a statement

'x is computable'

and denote it by the propositional variable X. (We shall use only lower case letters for denoting objects, so we can use the capital letters for the corresponding propositional variables here.)

Let, for any object t, $M(t)$ be its computational model and $M(t)_t^x$ is obtained by substituting X instead of T for every occurrence of T in the left-most position in names of propositional variables.

Let us define the following function called 'sem':

(1) For any primitive object t, $M(t)$ is empty.

(2) sem $(x{:}t) = M(t)^x_t$

(3) sem $(copy\ t) = M(t)$

(4) For any equation E
sem $(E) = \{X_1 \,\&\, \ldots \,\&\, X_{i-1} \,\&\, X_{i+1} \,\&\, \ldots \,\&\, X_m \rightarrow X_i \mid$ the equation E is solvable for $x_i\}$

(5) for any axiom A
sem $(A) = \{A\}$

(6) sem $(x{:}(S_1; \ldots; S_k)) = \{X \rightarrow X.Y, \ldots, X.Z, X.Y \,\&\, \ldots \,\&\, X.Z \rightarrow X\} \cup$
$\cup$ sem$(S_1) \cup \ldots \cup$ sem(S_k)

where $S_1, \ldots, S_k$ are of one of the following forms

— *copy c*,
— *y: s*,
— *relation*,

$Y, \ldots, Z$ are all propositional variables, associated with components of x.

It is easy to check that by applying the rules (1) to (6) for the function 'sem' as rewriting rules we can reduce any specification S to a set of axioms of the form (1), (2). This set is the computation model $M(S)$ of the specification S. This establishes the connection between the high-level knowledge representation language and IPC.

6. USING METATHEORIES

There is always a conflict between the generality of a calculus and the efficiency of derivation in it. The more specific the calculus we build, the better methods we can find for making inferences in this calculus. However, we also need some means of representing general knowledge, and this is a reason for coupling calculi of different generality. Another reason is more straightforward. Looking carefully at derivations in some calculus, we may discover some general laws of derivation which cannot be explicitly expressed in the calculus itself. This will lead to construction of a metatheory about the calculus, that is, a new calculus, objects of which are somehow related to the initial calculus, and which can help to improve the efficiency of search in the initial calculus.

We shall demonstrate the use of general knowledge with the following simple example. Let us have a concept of 'person', described in the knowledge representation language of the previous section as follows:

```
→person: (name: text;
          sex: text;
          age: numeric;
          father: person;
          mother: person)
```

Let us also have an intelligent data base of people where we can make inferences about persons and about sets of persons. In particular, we assume that this is done in IPC using the semantics described in the previous sections. Only two relatives, mother and father, are represented directly in the concept of a person. But we should like also to represent other relatives of a person, such as grandfather, uncle, mother-in-law, and many others. In order to define relationships of that kind in general, we need a first order calculus. Then we can write in a straightforward way:

father $(x,y) \rightarrow$ parent (x,y)
mother $(x,y) \rightarrow$ parent (x,y)
father (x,y) & parent $(y,z) \rightarrow$ grandfather (x,z)

and, for instance, having facts

father (John, Jack)
father (Jack, Jean)

can infer

grandfather (John, Jean).

Now we have two separate calculi for making inferences:

(1) a first order calculus for handling kinship relations;
(2) IPC for handling computability relations as described in section 5.

The question is: how do we couple them so as to use these two calculi jointly? In the NUT system [4] we have applied the schema shown in Figure 2. In addition to the usual semantics of specifications (arrow 1) there is another way, denoted by the arrows 2, 3, and 4. Specification is used as a source of facts. Using rules like those for grandfather, new facts are derived in Horn clause logic. These facts are represented as additional specifications in the high-level language and used further on as any other specification (arrows 5 and 6). The essential points are:

(1) specifications which are objects of the calculus 'sem' are sources of facts for the HCL;
(2) new facts derived in HCL give new specifications which give sets of axioms of IPC.

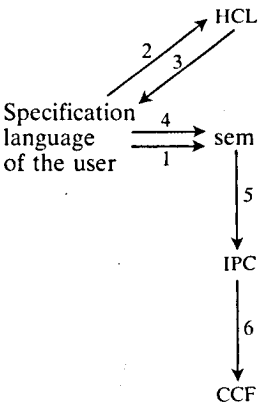


Figure 2. Calculi of NUT.

Let us demonstrate this on a problem where, given the following information:

Tom: person age = 55
John: person father = Tom;
Jane: person father = John;

we have to find Jane.grandfather.age. This can't be done in IPC and CCF. But we can solve the problem, using HCL to derive the new fact

grandfather (Tom, Jane)

which is further on transformed into the following specification

R001: grandfather Tom, Jane

and added to the specification of Tom, John, and Jane. This is how the system NUT uses HCL as a metacalculus for extending, firstly, specifications, and thereafter the set of axioms given in IPC.

For the example considered here we could also represent all the knowledge in a first order calculus, because the amount of knowledge is small. This is not the case in general. In particular, in CAD applications, thousands of axioms are needed for representing knowledge about computability [12], and the fragment of IPC which we use is a good choice due to the efficiency of inference.

7. DISCUSSION

What we have described in previous sections is the schema of connections of calculi used in the programming system NUT [4]. This schema is represented in Figure 2. A user supplies knowledge in a high-level knowledge representation language mostly for 'sem'. But the user must

give some knowledge also as programs in CCF and as rules in HCL. Knowledge is represented in the form of specifications of concepts. A specification of any concept can contain knowledge for the three calculi, and knowledge about a concept is stratified into three layers (see Figure 3).

(1) knowledge about structure and components, used directly in 'sem';
(2) knowledge about computability, used in IPC;
(3) general knowledge, used in HCL.

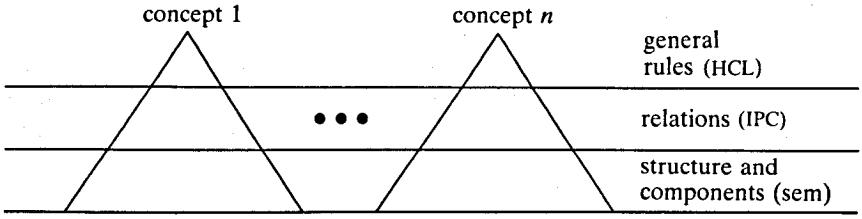


Figure 3. Stratification of knowledge.

A considerable amount of search is needed for making inferences in two calculi, HCL and IPC, whereas the calculi 'sem' and CCF can be used without backtracking. It appeared that dividing search between two calculi can be useful even when both calculi are propositional. This experience was gained from the system ExpertPRIZ [12]. The diagram of calculi of this system is represented in Figure 4. This system is a combination of an inductive expert system operating in a propositional calculus called ES, and a program synthesizer, containing the calculi 'sem', IPC, and CCF. This synthesizer operates in the same way as the program synthesis part of the NUT system. The effect is obtained because the ES calculus operates 'in the large': it makes short inferences, but gives large sets of new axioms for IPC. In this case, the ES is transparent to the user who can understand its logic and can control the inference. The calculi IPC and CCF operate on a lower level of the ExpertPRIZ system—on a level which is opaque to the user. There are only a few programmed functional constants in ExpertPRIZ and a user does not write any new programs. Implementations of axioms are derived from equations written in the knowledge representation language; this is denoted by the dotted arc from sem to CCF in Figure 4.

There are a number of other knowledge-based systems that combine several calculi. Truth maintenance systems (TMS) [13] and augmented truth maintenance systems (ATMS) [14] essentially use the idea of coupling two calculi. The basic idea of TMS and ATMS is to extend the capabilities of an automatic problem solver by coupling it with an automatic logician, TMS (or ATMS) which can do non-monotonic reasoning. The logician receives assumptions and facts from the solver and, using

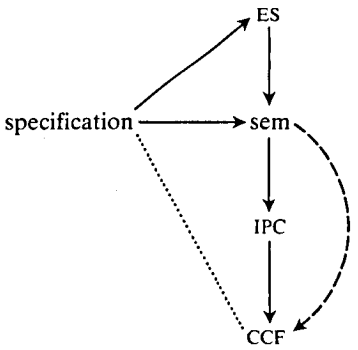


Figure 4. Calculi of ExpertPRIZ.

this knowledge, controls to some extent the behaviour of the solver, preventing it from inconsistency and useless search.

The extending of PROLOG is another area where attempts are made to unite several calculi. PROLOG itself unites HC and CCF very elegantly. But this elegance makes it difficult to add anything to PROLOG, in particular, new calculi. Any attempt to extend PROLOG substantially by adding a new calculus has led to eclectic solutions up to now. However, meta-computations seem to be a promising way for extending PROLOG. In particular, there is an interesting extension of PROLOG which enables one to use it in a way analogous to the usage of the PRIZ system [15]. But in this case PROLOG is used as an instrument, so that the calculi of PRIZ are simulated in PROLOG: HCL is used for simulating 'sem' and IPC.

The examples considered here have not covered the whole variety of knowledge-based systems which use more than one calculus each: it is a topical theme in expert systems research to unite several calculi. The aims of this paper have been:

(1) to draw attention to the fact that there are successful applications of the paradigm of the modularity of knowledge in the large by combining several knowledge systems;

(2) to outline a direction of research in the context of knowledge systems as formal calculi.

REFERENCES

1. Maslov, S. Yn. (1987). *Theory of deductive systems and its applications.* MIT Press, Cambridge, London.
2. Mints, G. and Tyugu, E. (1987). The programming system PRIZ. *J. Symbolic Computations*, No. 4.
3. Koov, M. *et al.* (eds.) (1986). *MicroPRIZ—Intelligent software systems.* Acad. Sc. of the ESSR, Tallinn.
4. Tyugu, E. *et al.* (1986). NUT—an object oriented language. *Computers and Artificial Intelligence*, **5**, No. 6, 521–42.

5. Moore, R. C. (1982). The role of logic in knowledge representation and commonsense reasoning. *Proc. of the AAAI* **82**, Pittsburgh, Pa, pp. 428–33.
6. Greif, I. and Hewitt, C. (1975). Actor semantics of PLANNER-73. *Proc. of 2nd ACM Symposium on Principles of Programming Languages*, NY, ACM, 67–77.
7. Nii, P. (1986). The blackboard model for problem solving. *Artificial Intelligence*, **7**, No. 2, 38–53.
8. Nii, P. (1986). Blackboard systems part two: blackboard application systems, *Artificial Intelligence*, **7**, No. 3.
9. Hayes-Roth, B. (1985). A blackboard architecture for control, *Artificial Intelligence*, **26**, 251–321.
10. Wirsing, M., Pepper, P., Partsch, H., Dosch, W. and Broy, M. (1983). On hierarchies of abstract data types. *Acta Informatica*, **20**, 1–33.
11. Mints, G. and Tyugu, E. (1986). Semantics of a declarative language. *Information Processing Letters*, **23**, 147–51.
12. Tyugu, E. (1987). Merging conceptual and expert knowledge in CAD, *Expert Systems in Computer-Aided Design*, North-Holland. pp. 423–34.
13. Doyle, J. (1979). A truth maintenance system, *Artificial Intelligence*, **12**, 231–72.
14. de Kleer, J. (1986). An assumption-based truth maintenance system, *Artificial Intelligence*, **28**, No. 2, 127–62.
15. Lomp, A. Computational models in PROLOG. This volume.

2

Propositional Logic Programming

G. Mints
Institute of Cybernetics,
Estonian Academy of Sciences, USSR

1. INTRODUCTION

We describe the application of propositional (mainly intuitionistic and modal) logic in logic programming. Logic programming is understood here in the broad sense of non-procedural programming in terms of logical specifications (Tyugu 1986), so that the compilation of the resulting program (i.e., program synthesis) is performed essentially by means of automatic proof search in a suitable logical system. We review the logical features of the programming system PRIZ developed at the Institute of Cybernetics of the Estonian Academy of Sciences. More detailed descriptions of the system architecture and proofs of some results can be found in Mints and Tyugu (1982, 1987).

The most familiar (and for many people the only) example of logic programming is Horn clause programming in first order predicate calculus which forms the logical base of the PROLOG language. Unfortunately *it is impossible to identify PROLOG* with Horn clause programming in view of numerous non-logical devices that are included in PROLOG in order to turn it into a viable programming language.

Propositional Horn clause logic is used in a number of program synthesis systems beyond PROLOG. The most developed of such systems known to the author is PRIZ. In fact the logical base of PRIZ is also more powerful than propositional Horn clause logic, but in another direction than in PROLOG: the planner (program synthesizer) of PRIZ is the complete procedure for the intuitionistic propositional calculus. The latter is known to be P-space complete, and one cannot expect good computational behaviour in the worst case. So as a programming short cut, independent subtasks have been introduced to speed up solutions of some problems arising in practice. It turned out to be complete for the corresponding fragment of the modal logic S4. Yet another short cut proposed in Kanovich (1985), but not implemented in PRIZ, turned out to be equivalent to the modal system S0.5

Another synthesis strategy in PRIZ permits the construction of *recursive programs.* Formalization of this strategy presented in Harf *et al.* (1983) has been obtained by introducing some syntactic restrictions to an *absolutely inconsistent system* (where any formula is derivable).

Something of this kind was to be expected since a recursive program can be partial, while any program synthesized according to the previous two strategies is (provably) total. In fact the basic rule of the recursive synthesis was found by suppressing the predicate structure in the familiar schema of transfinite induction

$$(\forall y)((\forall x < y)Ax \rightarrow Ay) \rightarrow (\forall x)Ax.$$

More details and examples of application of this strategy are presented in Mints and Tyugu (1982).

PRIZ combines conventional programming technique with automatic synthesis of programs from specifications. Its input language UTOPIST (Universal Translator Of Problems Including Specifying Texts) enables one to write specifications. Such a specification is automatically encoded into propositional calculus and used by the system for program synthesis.

Both PRIZ and PROLOG exploit the structural similarity of constructive proofs and programs and build a program by proving solvability of a problem. The PROLOG system works in first order predicate calculus and uses the resolution principle. Pure PROLOG handles objects of types zero and one (individuals and predicates). The logic of the PRIZ system is restricted to the propositional level. However implicitly it uses lambda calculus of finite type.

Today PRIZ is a program product installed on more than 1000 Ryad computer mainframes; it was originally developed as a practical programming system, and the Russian abbreviation is translated as 'programs for solving engineering problems'. Its logical background is invisible for a practically-minded user. From the user's point of view UTOPIST is essentially a non-procedural language.

We start our representation of the PRIZ system with a general description of its architecture in Section 2. Thereafter we describe the non-procedural part of its input language UTOPIST which is intended for writing specifications. In Section 3 we briefly discuss the logical basis of the PRIZ system, giving propositional semantics of specifications. Section 4 treats the structural synthesis of programs. An experiment with the synthesizer when all intuitionistic propositional theorems from Kleene (1952) have been proved is discusssed in Section 5; this section summarizes the paper Matskin *et al.* (1982). The last Section, 6, contains detailed treatment of the logical background of propositional program synthesis with independent subtasks.

Appendices contain formulation of the proof rules and program extraction rules of PRIZ as well as an example of the work of the system.

2. SYSTEM ARCHITECTURE AND INPUT LANGUAGE

Since automatic program synthesis is the main distinctive feature of

PRIZ, we present the system here mainly from that angle. The part of the PRIZ system we describe here is intended for processing *problem statements* of the form

$$M \vdash x_1, \ldots, x_k \rightarrow y \qquad (1)$$

which means 'knowing M compute y from $x_1, \ldots, x_k$', that is, it represents a computational problem. The following is an example of the problem statement:

triangle $\vdash a, b, c \rightarrow$ alpha.

Actually, PRIZ proves the solvability of the problem and from this proof derives a program, which calculates the value.

An essential part of the system is the knowledge base. It contains specifications of concepts and it is easily accessible by a user who can manipulate knowledge by adding specifications of new concepts and by editing the existing specifications.

The principal part of the PRIZ system is a synthesizer which translates a problem statement (1) into a program that performs the task described by this statement. Besides the problem statement, the synthesizer takes the following as input:

(1) the internal representation of the specification of M, which we call the *problem model*;

(2) programs and equations that realize the functional constants used in the problem model.

By proving solvability of the problem, the synthesizer builds the schema of a program for solving it. Thereafter, it assembles the program from solving functions of equations and program modules from the library. This program can be applied immediately for computations, or it can be used in a conventional programming system like any other program module.

The UTOPIST language appeared in 1974 as a problem specification language and it obtained its more or less final shape in 1977 (Kahro *et al.* 1981, Mints and Tyugu 1982). The specifications in UTOPIST represent *abstract objects* (concepts) which can be used for creating concrete objects (data structure): see examples below. Only concrete objects possess values. An abstract object is a carrier of information about the properties of concrete objects and in this sense it is analogical to a class in an object-oriented language.

The goal of a user is to specify an abstract object M, which enables the program to be represented by the problem statement (1).

We are going to illustrate by a series of examples the declarative non-procedural part of UTOPIST, that is the part which is used for specifying abstract objects. A more detailed and precise description is in the paper by E. Tyugu in this volume.

Simple examples of specifications are

point: (*x*: *numeric*;
y: *numeric*)

and

bar: (*P*1: point;
*P*2: point)

length: *numeric*;
angle: *numeric*;
length^2 = (*P*1.*x* − *P*2.*x*)^2 + (*P*1.*y* − *P*2.*y*)^2;
length**sin*(angle) = *P*2.*y* − *P*1.*y*

The two equations specify the properties of a bar operationally, by expressing *relations* of its components so that they can be used for computing the values of coordinates, the length, or an angle, depending on the problem statement.

A compound specifier (like point or bar) represents an object that can contain other objects which are then called its components. Compound names can be used for naming components of an object. A component *a* of an object *b* is called *b.a* outside of *b*. If *b*, in its turn, is the component of an object *c*, then outside of *c*, the name of the inner object is *c.b.a*, and so on.

A relation can be either an equation or an axiom with realization. In the case of an equation the system takes for granted that every variable occurring there can be computed from the remaining ones. There are various implementations for solving equations: numeric, symbolic, and also user-supplied.

A relation given by the axiom with realization has the form

$X \rightarrow Y(f)$

where *f* can be the name of a program from the program library computing *Y* from *X* (some details are given later),

Consider the following specification.

matrix: (*m*: *text*;
e: *numeric*;
i: *numeric*;
j: *numeric*;
create: → *m*(*A*);
put: *m*, *i*, *j*, *e* → *m*(*B*)
get: *m*, *i*, *j* → *e*(*C*));

This abstract object represents a matrix and it can be used as an abstract data type. Here *A*, *B*, and *C* are names of the programs which are respectively realizations of the relations 'create' (the matrix), 'put'

(an element e at the place i, j in the matrix m) and 'get' (an element from the place i, j in the matrix m).

A new object can be specified by the name of an abstract object followed by amendments which bind components of the object. For instance, having specifications of a point and a bar, we can write

P : point $x = 0$;
AB: bar length = 15, $P1 = P$;

The meaning of amendments $x = 0$ and length = 15, is obvious. The amendment $P1 = P$ in the specification of the bar AB means that the point $P1$ of the bar AB is the same as point P specified above.

Let us consider an example of a problem shown in Figure 1. The distance v must be computed depending on the value of the angle u. We can specify this problem as follows:

mech: (u : *numeric*;
v : *numeric*;
AB: bar length = 0.7, $P1 = (0, 0)$, angle = u;
BC: bar length = 1.5, $P1 = AB.P2$, $P2.y = -0.5$, $P2.x = v$)

The problem

mech $\vdash u \rightarrow v$

is solvable and the algorithm built by the PRIZ system where justification of each step is also shown is given in Appendix 3.

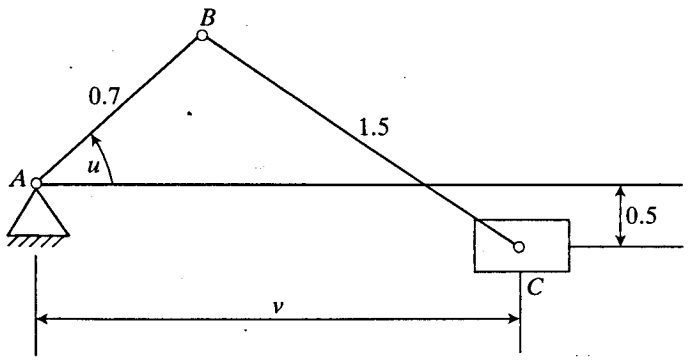


Figure 1. Problem to be solved by PRIZ system.

3. AXIOMATIC SEMANTICS OF SPECIFICATIONS

A precise representation of the semantics of UTOPIST can be given by means of a simple language which is a restricted (but still universal) form of the intuitionistic propositional calculus. The propositional variables X, Y, etc. express the computability (existence) of values of objects

presented by a specification. Let us denote the objects by small letters: a, b, x, y, a_1, a_2, ... For any object x we introduce a propositional variable X which denotes the computability of x. (X is true if x is computable or x already has a value.) The language includes only propositional formulas of the following forms:

$$X_1 \& \ldots \& X_k \rightarrow Y, \tag{2}$$

or in a shorter way:

$$\boldsymbol{X} \rightarrow Y;$$

as well as

$$(\boldsymbol{U}^1 \rightarrow V^1) \& \ldots \& (\boldsymbol{U}^m \rightarrow V^m) \rightarrow (\boldsymbol{X} \rightarrow Y), \tag{3}$$

or in a shorter way:

$$(\boldsymbol{U} \rightarrow V) \rightarrow (\boldsymbol{X} \rightarrow Y).$$

From the computational point of view these implications can be considered as functional dependencies. The formula (2) can be read simply as 'Y is computable from $X_1, \ldots, X_k$'. The formula (3) expresses functional dependency of higher order (with function as argument) and can be read 'Y can be computed from $\boldsymbol{X}$ and a function realizing $(\boldsymbol{U} \rightarrow V)$'.

To analyse the solvability of the computational problem given by a problem statement (1) and to find the applicative structure of the resulting program, only the purely propositional structure shown explicitly in (2), (3) is essential. However, to write the resulting program in all details, formulas (2), (3) have to be expanded as follows:

$$\boldsymbol{X} \xrightarrow{f} Y \tag{4}$$

and

$$(\boldsymbol{U} \xrightarrow{\boldsymbol{g}} V) \rightarrow (\boldsymbol{X} \xrightarrow{F(\boldsymbol{g})} Y), \tag{5}$$

where $\boldsymbol{g} = g_1, \ldots, g_m$.

Functions f, F in (4), (5) are realizations of (2), (3) respectively. The formula (4), for example, means that the realization of Y can be computed from the realizations of $\boldsymbol{X}$ by means of f, or that f is a procedure for computing y from $\boldsymbol{x}$. The formula (5) means that the procedure F produces from any functions $\boldsymbol{g}$ computing v from $\boldsymbol{u}$ some new function $F(\boldsymbol{g})$ computing y from $\boldsymbol{x}$.

Since any computational model consists of specifications, it is sufficient to define a function 'sem' which, for any specification S, computes a set sem(S) of formulas of the form (4), (5) which are axioms that describe the possible computations according to the specifications S. (A

computation with input variables $x_1, \ldots, x_k$, and an output variable y is possible according to the specification S if and only if the formula $X_1 \,\&\, \ldots \,\&\, X_k \rightarrow Y$ is constructively derivable from sem (S).) See the paper by Tyugu in this volume for details.

4. PROGRAM SYNTHESIS

The synthesizer of PRIZ employs the schema

$$\text{SPECIFICATION} \xrightarrow{\text{I}} \text{PROOF} \xrightarrow{\text{II}} \text{PROGRAM}. \qquad (6)$$

Input data for step I are produced by the function 'sem' mentioned above in the form of a sequent $\Gamma \vdash (P \rightarrow Q)$ with Γ (the axioms) being the list of propositional formulas (2), (3). The proof is a formal derivation of $P \rightarrow Q$ from Γ according to the so-called Structural Synthesis Rules (SSR) listed in Appendix 1. Its structure and the search strategy is best illustrated for the case when all axioms in Γ are of the form (2). Then one proceeds stepwise by gradually enlarging the set $\boldsymbol{C}$ of computed variables. Initially this set $\boldsymbol{C}$ for the goal sequent $\Gamma \vdash P \rightarrow Q$ consists of P (since its computability is assumed) and the variables given as separate members of Γ. Each search step simply adds to $\boldsymbol{C}$ all conclusions Y of a formula (2) if all premises $X_1, \ldots, X_k$ of this formula are already in $\boldsymbol{C}$. Then the formula (2) used in this way is simply discarded. The goal $\Gamma \vdash P \rightarrow Q$ is proved if Q is eventually included in $\boldsymbol{C}$. This proof search can be organized so that it is completed in linear time.

In the case when axioms of the form (3) are present in Γ, the proof search is more complicated and the resulting system turns out to be equivalent to the intuitionistic propositional calculus (Mints and Tyugu 1982).

Step II of the schema (6), that is, the extraction of the program from a constructed proof uses the same basic ideas as the standard intuitive interpretation of the intuitionistic system. Expanded versions (4), (5) of the formulas (2), (3) are used here to assign typed lambda-terms (realizations) to the axioms from Γ, which are starting formulas (leaves) of the proof (tree). Then we can proceed along the applications of the rules assigning realizations to further formulas. This assignment (see Appendix 2) uses a device traceable to the Heyting–Kolmogorov interpretation of intuitionistic connectives, or more precisely, the Kleene realizability (Kleene 1952). The lambda-term assigned to final formula $P \rightarrow Q$ is the schema of the required program.

To illustrate this, let us consider a specification for finding the row in a matrix, the maximal element of which has the minimal value among maximal elements of all rows. First of all we define a concept of 'maximum':

max: (
arg: *numeric*;
fun: *numeric*;
maxval: *numeric*;
(arg → fun) → maxval (*D*))

We use here a program *D* for representing a relation specified by the axiom

(arg → fun) → maxval.

This relation binds the maximal value of a function with the function itself represented by the subformula

arg → fun.

The concept of 'minimum' can be specified by using the same program *D*:

min: (
arg: *numeric*;
fun: *numeric*;
negfun: *numeric*;
minval: *numeric*;
maxval: *numeric*;
negfun = − fun;
minval = − maxval;
(arg → negfun) → maxval (*D*)).

And the specification of the desired concept 'minimax' is as follows:

minimax: (
value: *numeric*;
m: *matrix*;
*r*1: max arg = *m.j*, fun = *m.e*;
*r*2: min arg = *m.i*, fun = *r*1.maxval,
maxval = value).

The minimax problem can be represented in logical language by the following three axioms, where the propositional variables *M*, *I*, *J*, *E*, and MAXINROW denote computability of a matrix, of its number of row, column, element, and maximal element in a row. The variable MINIMAX denotes computability of the desired result of the problem.

$M\&I\&J \xrightarrow[\text{get}]{} E$

$(J \rightarrow E) \xrightarrow[\text{max}]{}$ MAXINROW

$(I \rightarrow \text{MAXINROW}) \xrightarrow[\text{min}]{}$ MINIMAX

These three axioms are a complete specification for synthesizing a

program that finds the minimal value of maximal elements of rows in a matrix.

The proof of solvability of this problem is

$$\dfrac{\dfrac{M\&I\&J\to E\ (J\to E)\to \text{MAXINROW}}{M\&I\to \text{MAXINROW}} \qquad (I\to \text{MAXINROW})\to \text{MINIMAX}}{M\to \text{MINIMAX}}$$

The complete program of this problem is

$$\lambda m.\min(\lambda i.\max(\lambda j.\text{get}(m,i,j))).$$

5. USE OF THE PROGRAM SYNTHESIZER AS A THEOREM PROVER

The program synthesizer (planner) of the programming system PRIZ has been used to produce natural deduction proofs of all intuitionistically derivable formulas from Kleene (1952) and to disprove underivable ones by exhaustive search for proof. This was made possible by the discovery (Mints and Tyugu 1982) that the planner is the complete proof search procedure for the conjunction–implication fragment of depth 2 of the intuitionistic propositional calculus, and could be extended to the whole calculus by adding a simple preprocessor working in quadratic time.

5.1. Preprocessor

Let us describe several transformations of the formulas of the intuitionistic propositional calculus (IPC) leading eventually to a form suitable for the planner of PRIZ and used in the preprocessor. The main ideas of these transformations can be traced to the logical works of the thirties (see Wajsberg 1938).

5.1.1. *Depth reduction*

New propositional variables are introduced for complex formulas to reduce the total depth. If the formula $F[A]$ contains complex formula A, then the transformation

$$F[A]\,|\to (X\leftrightarrow A)'\to F[X] \qquad (1)$$

is applied to move A outside. Here X is a new propositional variable which replaces A inside of F, and $(X\leftrightarrow A)'$ is an equivalent form of $(X\leftrightarrow A)$. More precisely,

$$
\begin{aligned}
(X \leftrightarrow A\&B)' &= (X \rightarrow A)\ \&\ (X \rightarrow B)\ \&\ (A \rightarrow (B \rightarrow X)) \\
(X \leftrightarrow AVB)' &= (A \rightarrow X)\ \&\ (B \rightarrow X)\ \&\ (X \rightarrow (AVB)) \\
(X \leftrightarrow A \rightarrow B)' &= (X \rightarrow (A \rightarrow B))\ \&\ ((A \rightarrow B) \rightarrow X) \\
(X \leftrightarrow (A \leftrightarrow B))' &= (X \rightarrow (A \rightarrow B))\ \&\ (X \rightarrow (B \rightarrow A))\ \& \\
&\quad ((A \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow X))
\end{aligned}
$$

5.1.2. *Elimination of negation and disjunction*

Negation $\tilde{}A$ is replaced by $(A \rightarrow L)$ after which L is eliminated by replacing the whole formula

$$F[L] \text{ by } (L \leftrightarrow K\&Z)' \rightarrow F[L]$$

where L and Z are new propositional variables and K is the conjunction of all propositional variables in F.

Up to this point all transformations increase the length of the formula linearly.

5.1.3. *Elimination of disjunction*

Previous transformations leave no L and disjunctions only in the form $(X \rightarrow BVC)$ to the left of the (main) arrow. Now replace

$$(X \rightarrow BVC)$$

by

$$\&_Z((B \rightarrow Z) \rightarrow ((C \rightarrow Z) \rightarrow Z)),$$

conjunction being taken over all variables Z in the whole formula.

The transformations just described enable one to put the formula (to be tested for provability) into the form

$$\Gamma \rightarrow p \tag{2}$$

where p is a propositional variable and Γ is a conjunction of formulas having one of the following forms

$$\begin{aligned}
&A1\ \&\ \ldots\ \&\ Ak \rightarrow B1\ \&\ \ldots\ \&\ Bl,\ k \geq 0,\ l \geq 1,\ Ai,Bj \\
&\text{propositional variables} \\
&R1\ \&\ \ldots\ \&\ Rm \rightarrow S1\ \&\ \ldots\ \&\ Sn,\ m,n > 0,\ Ri,Sj \\
&\text{being of the form (3).}
\end{aligned} \tag{3}$$

5.2. The experiment

Natural deductions found by the PRIZ planner are presented in so-called

Fitch form. Assumptions are marked by the symbol] (to be read *assume*). Consequences of an assumption are placed under it. Sometimes an indication (analysis) is placed at the right, explaining which passage rule was used. Recall that natural deduction of implication $A \rightarrow B$ by means of implication introduction rule ($\rightarrow^+$) proceeds by introduction of assumption A, deriving consequences and subsequent discharge of an assumption A when the goal B is achieved. This latter step is indicated by placing $A \rightarrow B$ below the whole derivation of B from A. Other natural deduction rules are elimination of implication ($\rightarrow^-$) and introduction and elimination of conjunction. Consider for example the derivation of the *sequent* $(A \rightarrow B),(B \rightarrow C),A \vdash C$ which is just another form of the formula $(A \rightarrow B)\&(B \rightarrow C)\&A \rightarrow C$.

1. $]A \rightarrow B$ (Assumption)
2. $]B \rightarrow C$ (Assumption)
3. $]A$ (Assumption)
4. B (from 1,3 by $\rightarrow^-$)
5. C (from 1,3 by $\rightarrow^-$)

5.2.1. *Examples of derivations constructed by computer*

The first example is reproduced directly from the computer printout, and the following ones given in a more readable form.

Example 1. $\vdash (A \rightarrow \sim\sim B) \leftrightarrow \sim\sim A \rightarrow \sim\sim B$ (Kleene 1952, *60I).

```
PRIZ-UTOPIST  Version 1.0  DATE 22.06.82  TIME 14.19.07
OPTIONS PACK, MACR,NOCMPR,NODBG,LIST,NOSKIP,ALGR
%
% EXAMPLE # 66                                    (*60I)
%
%  :— (A→ ~ ~B) = ~ ~A→B
%
    VAR: E,ABL,ALBL,BL,BLL,AL,A,B,Z;
    L:     CON  A,B,Z;
    BL:    IMP  B,L;
    BLL:   IMP  BL,L;
    AL:    IMP  A,L;
    ALL:   IMP  AL,L;
    ABL:   IMP  A,BLL;
    ALBL:  IMP  ALL,BLL;
    E:     EQ   ABL,ALBL;
    GOAL E;
***ALGORITHM***
    →
```

```
SUBPROBL
    →ABL
  SUBPROBL
      →ABL,ALL
    SUBPROBL
        →ABL,ALL,BL
      SUBPROBL
          →ABL,BL,A
            ABLIMP1:→BLL
            BLLIMP1:→L
      END OF SUBPROBL
          ALIMP2:→AL
          ALLIMP1:→L
    END OF SUBPROBL
        BLLIMP2:→BLL
  END OF SUBPROBL
      ALBLIMP2:→ALBL
END OF SUBPROBL
SUBPROBL
    →ALBL
  SUBPROBL
      →ALBL,A
    SUBPROBL
        →A,AL
          ALIMP1:→L
    END OF SUBPROBL
        ALLIMP2:→ALL
        ALBLIMP1:→BLL
  END OF SUBPROBL
      ABLIMP2:→ABL
END OF SUBPROBL
    EEQ3:→E
***END OF ALGORITHM***
```

Each of the following examples begins with the formula to be proved, then the result of its transformation into the sequent according to rules of the section 1 is presented, then the computer derivation (up to trivial permutations) is displayed. If a formula $(X \leftrightarrow E)'$ for a variable X is introduced in the process of the above-mentioned transformations, then the reference 'by X' means the inference by a rule applied to one of the conjunctive members of the conjunction $(X \leftrightarrow E)'$. The variable L is always the one introduced for elimination of the constant L according to 1.3. Reference 'by L' means inference by a rule applied to implication $L \rightarrow E$ for a variable E. Names of variables X in formulas $(X \leftrightarrow E)'$ were chosen to suggest the form of the formula E.

Example 2. $\neg\neg(\neg\neg A \rightarrow A)$ (Kleene 1952, *51B)

$L \leftrightarrow (A \& Z)$, $AL \leftrightarrow (A \rightarrow L)$, $AL2 \leftrightarrow (AL \rightarrow L)$, $DN \leftrightarrow (AL2 \rightarrow A)$, $DN \rightarrow L \vdash L$

1.	$DN \rightarrow L$	
2.	$\quad]A$	(assumption)
3.	$\quad\quad]AL2$	(assumption)
4.	$\quad\quad A$	(from 2)
5.	$\quad AL2 \rightarrow A$	(from 4 by $\rightarrow^+$)
6.	$\quad DN$	(from 5 by DN)
7.	$\quad L$	(from 1, 6 by $\rightarrow^-$)
8.	$A \rightarrow L$	(from 7 by $\rightarrow^+$)
9.	AL	(from 8 by AL)
10.	$\quad]AL2$	(assumption)
11.	$\quad L$	(from 9, 10 by $AL2$)
12.	$\quad A$	(from 11 by L)
13.	$AL2 \rightarrow A$	(from 12 by $\rightarrow^+$)
14.	DN	(from 13 by DN)
15.	L	(from 1, 14 by $\rightarrow^-$)

Example 3. $\neg B \vdash (AVB) \leftrightarrow A$ (Kleene 1952, *48)

$L \leftrightarrow (A \& B \& Z)$, $D \leftrightarrow (AVB)$, $E \leftrightarrow (D \leftrightarrow A)$, $B \rightarrow L \vdash E$

1.	$B \rightarrow L$	
2.	$\quad]D$	(assumption)
3.	$\quad\quad]A$	(assumption)
4.	$\quad\quad E$	(by E)
5.	$\quad A \rightarrow E$	
6.	$\quad\quad]B$	(assumption)
7.	$\quad\quad L$	(from 1)
8.	$\quad\quad E$	(by L)
9.	$\quad B \rightarrow E$	
10.	$\quad E$	(by D)
11.	$\quad A$	(from 2, 10 by E)
12.	$D \rightarrow A$	
13.	$\quad\quad]A$	(assumption)
14.	$\quad\quad D$	(from 13 by D)
15.	$A \rightarrow D$	
16.	E	(from 12, 15 by E)

Example 4. $(A \rightarrow \neg B) \leftrightarrow \neg(A \& B)$ (Kleene 1952, *58B)

$L \leftrightarrow (A \& B \& Z)$, $BL \leftrightarrow (B \rightarrow L)$, $K \leftrightarrow (A \& B)$, $KL \leftrightarrow (K \rightarrow L)$, $ABL \leftrightarrow (A \rightarrow BL)$, $E \leftrightarrow (ABL \leftrightarrow KL) \vdash E$

1.	$\quad\quad]ABL$	(assumption)
2.	$\quad\quad\quad]K$	(assumption)

3.		A	(by K)
4.		B	(by K)
5.		BL	(from 1, 3 by ABL)
6.		L	(from 4, 5 by BL)
7.	$K \rightarrow L$		
8.	KL		(from 7 by KL)
9.	$ABL \rightarrow KL$		
10.	$]KL$		(assumption)
11.		$]A$	(assumption)
12.			$]B$ (assumption)
13.			K (from 11, 12 by K)
14.			L (from 10, 13 by K)
15.	$B \rightarrow L$		
16.	BL		(by BL)
17.	$A \rightarrow BL$		
18.	ABL		(by ABL)
19.	$KL \rightarrow ABL$		
20.	E		(by E)

6. INDEPENDENT SUBTASKS AND THE MODAL LOGIC S4

It had been noted already that planning with dependent sub-tasks (that is the general planning strategy of PRIZ) provides sound and complete proof procedure for the intuitionistic propositional calculus IPC. For problems with few subtasks (that is, nested implications, see below) its execution is finished in feasible time. The general decision problem for the intuitionistic propositional calculus was proved to be P-space-complete (Ladner 1977), so it would be unreasonable to expect good computational behaviour of the planner (proof search unit) of PRIZ in the worst case. The authors of PRIZ, who did not know these general considerations, noticed that the application of the general algorithm leads to long and useless search during solution of some problems that interested them. They introduced (from heuristical considerations, see Tyugu and Harf (1980)) another planning strategy which they called the mode of independent subtasks. It is determined by the proviso: different subtasks cannot help each other. More precisely, two computability statements

$$(A \rightarrow B) \,\&\, C \rightarrow D, (A' \rightarrow B') \,\&\, C' \rightarrow D'$$

can help each other only via their conclusions D, D', and not via premises A,A' of their subtasks.

We shall prove that planning with independent subtasks leads to a sound and correct decision algorithm for the modal logic $S4$, or more

precisely for its fragment where only computability statements are considered, and $\rightarrow$ is understood as strict implication (see below). Basic differences from the intuitionistic case is that for $S4$ these formulas most probably do not form the reduction class.

It will be convenient to write unconditional computability statements (2), section 3 in the form $K \rightarrow V$ (where K is a conjunction of propositional variables and V is a variable) and conditional computability statements (3), section 3 in the form $R_1 \,\&\, \ldots \,\&\, R_n \rightarrow V$, where $R_1, \ldots, R_n$ are unconditional computability statements.

The specification of the program is of the form

$$\text{On } (R_1, \ldots, R_n) \text{ from } V_1, \ldots, V_m \text{ compute } V.$$

It is translated by the propositional formula

$$(R_1 \,\&\, \ldots \,\&\, R_n \,\&\, V_1 \,\&\, \ldots \,\&\, V_m \rightarrow V) \tag{1}$$

The rules of structural synthesis with independent subtasks can be represented in the following way in terms of sequents $\Gamma \rightarrow A$, so that formula (1) is translated by a sequent

$$R_1, \ldots, R_n, V_1, \ldots, V_m \rightarrow V.$$

Let us denote by SSRind the system which is determined by the axiom schema $\Gamma, V \rightarrow V$ and the following two rules:

$$\frac{\displaystyle\mathop{\&}_{1 \leqslant i \leqslant p} (\boldsymbol{K}_i \rightarrow U_i) \,\&\, \mathop{\&}_{1 \leqslant j \leqslant q} V_j \rightarrow V;\ \tilde{\boldsymbol{K}}_i \vdash U_i\ (1 \leqslant i \leqslant p);\ Y_j \vdash V_j\ (1 \leqslant j \leqslant q)}{Y_1, \ldots, Y_q \vdash V}$$

which is denoted by $(\dot{\rightarrow}--)$ind, where $\tilde{\boldsymbol{K}}$ is any list consisting of variables occurring in the conjuction $\boldsymbol{K}$ (not necessarily all of them) and

$$\frac{\displaystyle\mathop{\&}_{1 \leqslant j \leqslant q} V_j \rightarrow V;\ Y_j \vdash V_j\ (1 \leqslant j \leqslant q)}{Y_1, \ldots, Y_q \vdash V} \quad (\rightarrow -)$$

This system is suitable for deriving sequents

$$V_1, \ldots, V_m \vdash V \tag{2}$$

from assumptions. The basic difference from the rule $(\rightarrow - -)$ of the system PRIZ is in the second premise of the rule $(\rightarrow - -)$ind: now it does not contain antecedent formulas of the conclusion, and it was through them that other subtasks could help.

Consider *translation* of formulas and sequents in the language of modal logic which sends implications $(A \rightarrow B)$ into strict implications $\Box(A \supset B)$. The translation of the expression E will be denoted by E'.

Our main aim in this section is the proof of the following statement:

Theorem 1. The sequent

$$R'_1, \ldots, R'_n, V_1, \ldots, V_m \vdash V \tag{3}$$

is derivable in $S4$ if and only if the sequent (2) is derivable from assumptions $R_1, \ldots, R_n$ in ssrind.

Proof: recall that the rules of the Gentzen-type sequent version of $S4$ have the form

$$\frac{A, \Box A, X \to Y}{\Box A, X \to Y}\ (\Box\vdash) \qquad \frac{\Box X \to A}{Y, \Box X \to \Box A}\ (\vdash\Box)$$

$$\frac{X \vdash Y, A;\ B, X \vdash Y}{(A \supset B), X \vdash Y}\ (\supset\vdash) \qquad \frac{A, X \vdash Y, B}{X \vdash (A \supset B), Y}\ (\vdash\supset)$$

$$\frac{A, B, X \vdash Y}{(A \& B), X \vdash Y}\ (\&\vdash) \qquad \frac{X \vdash Y, A;\ X \vdash Y, B}{X \vdash Y, (A \& B)}\ (\vdash\&)$$

Let us prove first the correctness of our translation. Let a derivation d of the sequent (2) from assumptions $R_1, \ldots, R_n$ in the system ssrind be given. Add formulas $R_1, \ldots, R_n$ to the left of $\vdash$ sign in all sequents occurring in d, and replace all formulas of the form $(A \to B)$ by $\Box(A \supset B)$. Then the axioms $X, V \vdash V$ become axioms $Y, X, V \vdash V$, assumptions R_i become axioms $R'_1, \ldots, R'_n \vdash R'_i$, and any inference according to the rules $(\to\text{ -- }\text{-})$ind, $(\to\text{-})$ becomes a sequence of inferences according to the rules of $S4$. So the sequent (3) is derivable in $S4$, exactly as required.

To prove that our translation is faithful, we shall apply a specialization of derivations in $S4$ familiar from proof theory.

Lemma: Every derivation of the sequent (3) in $S4$ can be transformed into a one-succedent derivation of the same sequent (any sequent in the derivation contains no more than one formula to the right of $\vdash$) and any $(\Box\vdash)$-inference is contained in a figure of one of the following forms:

$$\cfrac{\cfrac{\ldots \Box I, Y \vdash V_j \ldots}{\Box I, Y \vdash \& V_j;} \qquad V, \Box I, Y \vdash W}{\cfrac{I, \Box I, Y \vdash W}{\Box(\ \&\ \ V_j \supset V), Y \vdash W}} \qquad (4)$$
$$1 \leqslant j \leqslant q$$

where V_i, W are variables and I denotes $(\ \&\ \ V_j \supset V)$
$1 \leqslant j \leqslant q$

or

$$\dfrac{\dfrac{\dfrac{\Box I, Y^-, \tilde{K}_i \vdash U_i}{\Box I, Y^-, K_i \vdash U_i}}{\Box I, Y^- \vdash (K_i \supset U_i)}}{\ldots \Box I, Y \vdash \Box(K_i \supset U_i) \ldots; \ldots \Box I, Y \vdash V_j \ldots} \qquad (5.1),\ (5)$$

$$\dfrac{\Box I, Y \vdash \underset{i}{\&}\, \Box(K_i \supset U_i) \,\&\, \underset{j}{\&}\, V_j; \qquad \Box I, V, Y \vdash W}{\vdash \Big(\underset{1 \leqslant i \leqslant p}{\&}\, \Box(K_i \supset U_i) \,\&\, \underset{1 \leqslant j \leqslant q}{\&}\, V_j \supset V\Big), Y \vdash W} \qquad (5.2)$$

where I denotes $\Big(\underset{1 \leqslant i \leqslant p}{\&}\, \Box(K_i \supset U_i) \,\&\, \underset{1 \leqslant j \leqslant q}{\&}\, V_j \supset V\Big)$, the $U_i\,(1 \leqslant i \leqslant p)$, V_j $(1 \leqslant j \leqslant q)$, V, W are variables, K_i $(1 \leqslant i \leqslant p)$. are conjunctions of variables, Y^- denotes the result of deleting from the list Y all its elements which do not belong with $\Box$, and $\tilde{K}$ is the list obtained from a conjunction K by replacing all & by commas; any $(\vdash\Box)$-inferences are contained in figures of the form (5).

The proof of the lemma: consider an arbitrary cutfree derivation of the sequent (3) in the system $S4$. In view of the subformula property the main formula of any $(\Box\vdash)$-inference is of the form $\Box I$ as in (4) or (5), and the main formulas of $(\vdash\Box)$-inferences are of the form $\Box(K_i \supset U_i)$ and originate from formulas $\Box I$ situated to the left of $\vdash$. Using *invertibility* of all rules for the propositional connectives &, $\supset$, we can ensure that all $(\Box\vdash)$-inferences introducing formulas containing a unique occurrence of $\Box$ (i.e., translations of the unconditional computability statements) are contained in figures which differ from (4) at most in one respect: W can be a list of variables and of formulas of the form $\Box(K \supset U)$, and W is added to the left of $\vdash$ in two upper sequents. $(\Box\vdash)$-inferences introducing translations of the conditional computability statements can be assumed to be inside figures which differ from (5) in a similar, inessential way. Similarly, $(\vdash\Box)$- and $(\vdash\supset)$-inferences are contained in figures of the form

$$\dfrac{\dfrac{\dfrac{X^-, \tilde{K}_i \vdash U_i}{X^-, K_i \vdash U_i}}{X^- \vdash K_i \supset U_i}}{X \vdash \Box(K_i \supset U_i), Z} \qquad (6)$$

which are similar to the figure (5.1), where Z consists of variables and $\Box$-formulas.

Let us now delete, beginning from the top, all superfluous formulas.

This is first done for axioms, and then for other sequents. All inferences and whole branches of the derivation tree which become superfluous in the process are deleted too. Let us etablish that exactly one formula will remain to the right of $\vdash$ in every sequent. This property is obvious for axioms and is preserved by all inferences except $\vdash\supset$. But in our case it is satisfied even for $(\vdash\supset)$, since it is contained in (6). To finish the proof it remains only to verify that the parts (5.1) and (5.2) of the figure (5) cannot be situated too far away and that the formula W in (4), (5) cannot be of the form $\Box(K\supset U)$. Suppose for contradiction that some figure (5.1) is not immediately above a corresponding figure (5.2). Choose the lowermost such figure Φ. Immediately above it some figure of the form (4), (5) or (5.1) should be situated, and Φ is situated immediately above the right premise of the rule $(\vdash)$. Assume, to simplify the notation, that it is (4) that is situated under Φ. Then the relevant part of the derivation is of the form.

$$\dfrac{\dfrac{\dfrac{

In our case there was no necessity to use the complicated restriction on the natural deduction in $S4$ imposed by Prawitz (1965) or to introduce the apparatus of the square brackets from Mints (1974), since implications of the form $A \supset \Box B$ are not allowed in our language. It is to avoid such formulas that we expressed specifications in the form (1) instead of intutionistically (but not $S4-$) equivalent multiple implication.

APPENDIX 1

The inference rules for structural synthesis of programs (SSR)

$$\frac{\vdash \boldsymbol{X} \to V; \Gamma \vdash \boldsymbol{X}}{\Gamma \vdash V}, (\to -)$$

where $\Gamma \vdash \boldsymbol{X}$ is a set of sequents for all X in $\boldsymbol{X}$.

$$\frac{\vdash (\boldsymbol{U} \to V) \to (X \to Y); \Gamma \vdash \boldsymbol{X}; Z, \boldsymbol{U} \vdash V}{\Gamma Z \vdash Y} (\to - -)$$

where $\Gamma \vdash \boldsymbol{X}$ is a set of sequents for all X in $\boldsymbol{X}$, and $Z, \boldsymbol{U} \vdash V$ is a set of sequents for all $(U \vdash V)$, in $(\boldsymbol{U} \vdash V)$.

$$\frac{\Gamma, \boldsymbol{X} \vdash Y}{\Gamma \vdash \boldsymbol{X} \to Y} (\to +)$$

In fact, the planner of PRIZ uses some additional rules which can be derived from the basic ones listed above. For example in the rule $(\to - -)$ the rightmost $\boldsymbol{U}$ above the line can be replaced by $\boldsymbol{W} \& \boldsymbol{U}$, and $\boldsymbol{W}$ added below the line.

APPENDIX 2

We present here program derivation rules. Taking into account

$$X \underset{f}{\to} Y = (\forall s)\,(X(s) \to Y(f(s))$$

and

$$(U \underset{g}{\to} V) \to (X \underset{F(g)}{\to} Y) =$$
$$(\forall g)\,((\forall u)\,(U(u) \to V(g(u))) \to (\forall x)\,(X(x) \to (Y(F(g,x))),$$

we can extend the inference rules SSR so that they will contain the rules for building new terms:

$$\frac{\vdash X \underset{f}{\to} V; \Gamma \vdash X(t)}{\Gamma \vdash V(f(t))} (\to -)$$

$$\frac{\vdash (U \underset{g}{\rightarrow} V) \rightarrow (X \underset{F(g)}{\longrightarrow} Y); \Gamma \vdash X(s); Z, U \vdash V(t)}{\Gamma, Z \vdash Y(F(\lambda u.t)\ (s))} (\rightarrow - -)$$

$$\frac{\Gamma, X \vdash Y(t)}{\Gamma \vdash X \underset{\lambda x.t}{\longrightarrow} Y} (\rightarrow -)$$

These rules represent the method for constructing a program simultaneously with the proof.

APPENDIX 3

Computer printout of the algorithm for the problem mech.

```
AB.l = 0.7 → AB.l
AB.P1.x = 0 → AB.P1.x
AB.P1.y = 0 → AB.P1.y
BC.l = 1.5 → BC.l
BC.P2.y = − (0.5) → BC.P2.y
→ u
AB.angle = u → AB.angle
sin(AB.angle)*AB.l = AB.P2.y − AB.P1.y → AB.P2.y
BC.P1.y = AB.P2.y → BC.P1.y
AB.l*AB.l = (AB.P2.x − AB.P1.x)^2 + (AB.P2.y − AB.P1.y)^2
  → AB.P2.x
BC.P1.x = AB.P2.x → BC.P1.x
BC.l*BC.l = (BC.P2.x − BC.P1.x)^2 + (BC.P2.y − BC.P1.y)^2
  → BC.P2.x
BC.P2.x = v → v
***end of algorithm***
```

REFERENCES AND BIBLIOGRAPHY

Clocksin, W. and Mellish, C. (1981). *Programming in Prolog.* Springer-Verlag, Berlin.

Curry, H. B. (1963). *Foundations of Mathematical Logic.* McGraw-Hill.

Gabbay, D. M. (1986). Negation as inconsistency. I. *J. Log. Progr.*, **1**, 1–35.

Gabbay, D. M. and Reyle, U. (1984–85). N-Prolog, and extension of Prolog with hypothetical reasoning. Part I. *J. Log. Progr.* (1984) **1**, 319–55; Part II. Ibid. (1985), **2**, 251–84.

Harf, M., Mints, G., Penyam, J., and Tyugu, E. (1983). Structural synthesis of recursive programs. In: *Automatic synthesis of programs.* Inst. of Cybern., Estonian Academy of Science. Tallinn, pp. 58–70 (Russian).

Kahro, M., Kalja, A., and Tyugu, E. (1981). *Instrumental programming system ES EVM (PRIZ)* (Russian). Moscow, Finansy i Statistika.

Kanovich, M. I. (1985). Lossless calculi in the effective schematic synthesis of programs (Russian). Soviet Conference in *Applied Logic.* Novosibirsk pp. 98–100.

Kleene, S. (1952). *Introduction to metamathematics.* North-Holland, Amsterdam.

Kowalski, R. (1979). *Logic for problem solving.* North-Holland, Amsterdam.
Ladner, R. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.* **6**, No. 3, 467–80.
Matskin, M., Mints, G., Tyugu, E., and Volozh, B. (1982). Theorem proving with the aid of program synthesizer. *Cybernetics*, No. 6, 63–70.
Mints, G. (1974). Lewis' systems and the system T. (Russian). In: *Feys R., Modal Logic.* Moscow, Nauka, pp. 422–509.
Mints, G. (1984). Structural synthesis with independent subtasks and the modal logic S4. *Proc. Estonian Academy of Sci., Ser. Mathem.*, **33**, No. 2, 147–51 (Russian).
Mints, G. (1986). Complete calculus for the pure Prolog. (Russian) *Proc. Estonian Academy of Sci.*, (1987), **35**, no. 4.
Mints G. and Tyugu, E. (1982). Justification of the structural synthesis of programs. *Sci. of Computer Progr.*, no. 2, 215–40.
Mints, G. and Tyugu, E. (1987). The programming system PRIZ. *Journal of Symbolic Computations*, No. 4.
Penyam, J. (1983). The synthesis of a semantic processor from attribute grammar. *Soviet Computer Programming*, No. 2. 50–62 (Russian).
Prawitz, D. (1965). *Natural deduction.* Almquist and Wiksell, Stockholm.
Schütte, K. (1977). *Beweistheorie.* Springer-Verlag, Berlin.
Tyugu, E. (1986). The future of logic programming. *Information Processing 86* (ed. H. J. Kugler). North-Holland, pp. 225–7.
Tyugu, E. and Harf, M. (1980). Algorithms of the structural synthesis of programs. (Russian). *Computer Programming*, no. 4, 3–13.
Vorobyev, N. N. (1958). A new derivability algorithm in the constructive propositional calculus. *Proc. Steklov Math. Inst.*, **52**, 193–226. (Russian, English translation by AMS).
Wajsberg, M. (1938). Untersuchungen über den Aussagenkalkül von A. Heyting, *Wiadomosci Matematyczne*, **46**, 45–101.

3

Computational Models in PROLOG

A. A. Lomp
Institute of Cybernetics,
Tallinn, USSR

1. INTRODUCTION

The programming language PROLOG enables us to write declarative programs where the order of predicates is irrelevant to the correctness and functioning of the program.

On the other hand, undoubtedly, some computational programs containing an evaluable predicate are not declarative as the evaluable predicates have restrictions in calling them (for example, the predicate 'X is Y*Z' can be called only when the variables 'Y' and 'Z' are bound).

Nevertheless, it is possible to write declarative computational programs using special programming disciplines. One of them was developed at the Institute of Cybernetics by Enn Tyugu and implemented in the programming system PRIZ—it is a formalism of computational models [1, 2]. In this paper we will study examples of the computational model approach, and discuss the discipline for solving computational problems stated in terms of computational models in PROLOG.

2. WHAT IS A COMPUTATIONAL MODEL?

A *computational model* represents the meaning of a problem in terms of computability. It is a production system where the situation is determined by a set of variables with computable values and the actions are determined by a set of computational rules with calculating programs. These computational rules define when calculating programs are applicable.

For example, the meaning of a geometrical object 'square' can be represented by three variables and two rules:

K = length of side,
D = length of diagonal,
S = area of square,
if nonvar(K)&var(D) or nonvar(D)&var(K)
  then equ($D = K$*sqrt(2)),
if nonvar(K)&var(S) or nonvar(S)&var(K)
  then equ($S = K\hat{}\ 2$),

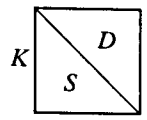


Figure 1.

where

nonvar—tests whether variable is currently bound,
var —tests whether variable is currently unbound,
equ —solves equation.

The *calculating programs* called from the computational model have input variables (to be computed before execution) and output variables (to be computed after execution).

In the first rule, the call 'equ($S = K\hat{}2$)' has the input variable K and the output variable S or the input variable S and the output variable K.

The *computational rules* have the special form: 'if the input variables have been computed and the output variables have not been computed then the calculating program is executed'.

In the example, the formula 'nonvar(K)&var(S) or nonvar(S)&var(K)' tests whether the input variable K is bound and the output variable S is unbound or the input variable S is bound and the output variable K is unbound; the call 'equ($S = K\hat{}2$)' applies a program 'equ' solving the equation '$S = K\hat{}2$'.

The computational model can be used for calculating the values of the variables. The algorithm for calculating is actually a forward reasoning algorithm which scans the production rules until new values are computed.

The computational model 'square' can be used for calculating variables K, D, and S. For example, if the value of S is obtained then the value of D would be computed by scanning the computational rules twice (the first time K is computed using rule 2 and the second time D is computed using rule 1).

3. HOW IS A COMPUTATIONAL MODEL WRITTEN?

A computational model can be encoded as a program memorizing all values and scanning all computational rules.

The computational model 'square' can be written as one clause that memorizes all variables (encoded as arguments) and scans all rules (encoded as successful procedures '→').

```
square(K,D,S):—
  weak([K,D]) → equ(D = K*sqrt(2)),
  weak([K,S]) → equ(S = K^2).
```

where

```
weak([X,Y]):—          % predicate 'weak([X,Y])' tests
  nonvar(X),var(Y);    % the condition 'nonvar(X)&var(Y) . . .'
  nonvar(Y),var(X).
```

```
C→A:—
  call(C),          % procedure 'C→A' encodes
  call(A);          % the rule 'if C then A'
  true.
```

The program 'square' can be simplified using a special procedure 'rel' that generates a list of variables (has_variable($S = K\hat{}2$,[K,S])), tests the condition (weak([K,S])) and applies the program-solving equations (equ($S = K\hat{}2$)).

```
square(K,D,S):—
  rel(D = K*sqrt(2)),
  rel(S = K^2).
```

where

```
rel(Equation):—
  has_variables(Equation,List),
  weak(List) → equ(Equation).
```

A computational model can be encoded hierarchically. Using the already encoded computational models we can define new computational models. In the next example an object 'figure' contains the object 'square'. And the definition of the model 'figure' contains the definition of the model 'square'.

```
figure(K1,D1,S1,D1,D2,S2,S):—
  square(K1,D1,S1),
  square(D1,D2,S2),
  rel(S = S2 − S1/2).
square(K,D,S):—
  rel(D = K*sqrt(2)),
  rel(S = K^2).
```

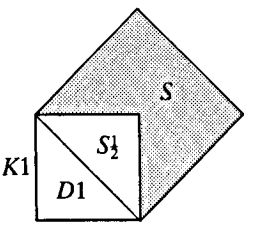


Figure 2.

To abstract from the unnecessary arguments in the clause 'figure' we include a variable Q in the clause 'square'. After that a new procedure 'figure' has three arguments: $Q1$, $Q2$ and S. Nevertheless the new procedure memorizes all variables of the computational model, as $Q1$ includes $K1$, $D1$, $S1$ ($Q1 = [K1,D1,S1]$) and $Q2$ includes $K2$, $D2$, $S2$ ($Q2 = [K2,D2,S2]$).

```
figure(Q1,Q2,S):—
  square(Q1,_,D1,S1),
  square(Q2,D1,_,S2),
  rel(S = S2 − S1/2).
square(Q,K,D,S):—
  Q = [K,D,S],
  rel(D = K*sqrt(2)),
  rel(S = K^2).
```

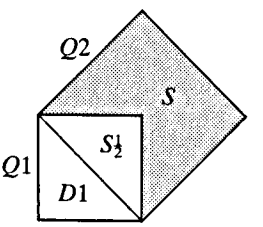


Figure 3.

In the formalism of computational models it is assumed that the order of rules is irrelevant (it does not matter for the functioning of the program). Consequently any computational model is definable by a declarative program where, in any clause, the order of predicates is immaterial. In other words, a programmer can change the order of computational rules (for example, he can change the order of 'rel($D = K$*sqrt(2))' and 'rel($S = K\hat{}2$)'.

4. HOW TO USE A COMPUTATIONAL MODEL

A computational model is used to compute all variables or to compute a particular variable or variables.

To compute all variables the computational model is called several times (one call executes all computational rules).

In the following example the value of S is one. The first call computes K; the second, D; the last, nothing.

```
?-square(_,K,D,1),square(_,K,D,1),square(_,K,D,1).
K = 1,
D = 1.4142
yes
```

The computational model must be called until something is computed. This can be done by a special procedure 'com' that calls the computational model until no rule is fired (new values are not computed by any 'rel').

```
?-com(square(_,K,D,1)).
K = 1
D = 1.4142
yes
```

To compute particular variables all values are computed and particular values are tested. In the next query the variable D is computed and tested.

```
?-com(square(_,_,D,1)),nonvar(D).
D = 1.4142
yes
```

If we have a special procedure 'pro' that computes all variables from input variables and tests values of output variables, then the variable D can be computed by a query.

```
?-S = 1,pro(square(_,K,D,S),[D],[S]).
D = 1.4142
yes
```

where

[*D*] is a list of output variables (to be computed after call),
[*S*] is a list of input variables (to be computed before call).

The procedure 'pro' can be called from any calculating program. For example, it can be called from a program 'sum_up' that has an input variable *A* and an output variable *Z*.

```
sum_up(0,Z):-                    % The computational
  pro(sum(_,_,Z,0),[Z],[0]).     % program 'sum_up'
sum_up(A,Z):-
  pro(sum(_,_,Y,A),[Z],[A]),
  A1 is A-1,
  sum_up(A1,Z1),
  Z is Z+Z1.
```

The program 'sum_up' and the program 'is' are called from the computational model 'sum'.

```
sum(A,Z,Y,X):-                   % The computational
  (nonvar(X),var(Y)→Y is X^2),   % model 'sum'
  (nonvar(A),var(Z)→sum_up(A,Z)).
```

The procedure 'sum_up' tests the predicate 'pro'. To test this predicate before calling the procedure we would rewrite the computational model 'sum'.

```
sum(A,Z,Y,X):-
  (nonvar(X),var(Y)→Y is X^2),
  (nonvar(A),var(Z),pro(sum(_,_,Y,X),[Y],[X])→sum_up(A,Z)).
```

That is to say, we have extended the computational rules with the predicate 'pro'. This is true when its output variables are computable from its input variables using a computational model. Otherwise this predicate is false.

5. IMPLEMENTATION

To write declarative computational programs we have extended the programming system PROLOG with new system-defined procedures (rel,com,pro,...).

These procedures are executed by the PROLOG interpreter. But they can also be executed by a special meta-interpreter that synthesizes the text of a program.

```
?-pro(square(K,D,S),[S],[D]).
```

```
square(K,D,S):-                    % The text of the program
  equ(D = K*sqrt(2)),              % synthesized by the
  equ(S = K^ 2).                   % metainterpreter
yes
```

The meta-interpreter and added procedures have been implemented in micro-PROLOG [3] and in CPROLOG [4] (the system micro-PROLOG runs under MSDOS on the IBM PC and the system CPROLOG runs under UNIX on SUN).

6. APPLICATIONS

The present system and the system PRIZ are based on the same formalism—the computational model. So from the theoretical point of view they can be used for solving the same problems.

The system PRIZ has been used for simulating power semiconductor thyristors, for designing shafts, for computing active filters, for analysis of gear transmissions, and so on. The present system can solve all these problems, but its current use is for computing geometrical objects, parameters of electrical circuits, characteristics of hydraulic drives, and for simulating logic cells.

Furthermore, in presenting this formalism we defined the computational model 'square' that has two rules. The systems above have been efficiently used to solve problems where the model contains more than 10,000 rules.

7. CONCLUSION

In this paper we claim that computational programs can be written as specifications if we carefully follow the discripline of programming computational models in PROLOG.

Unlike many disciplines for writing specifications, this gives relatively effective computational programs that can be used in practice.

Acknowledgements

I am grateful to the Institute of Technical Cybernetics in Czechoslovakia and to the Turing Institute in Scotland for technical support when completing this research. I greatly benefited from comments, suggestions, and discussions from Enn Tyugu, Grigory Mints, Leo Motus and Peeter Lorents. I would also like to thank Donald Michie for his encouragement and Guy Narbony who implemented the equation-solver.

APPENDIX—COMPUTING PARAMETERS OF ELECTRICAL CIRCUITS

Here follows a package for analysis of alternating-current electrical circuits made of capacitors and inductors. The class of circuits is restricted

to parallel-series connection of ports characterized by current, voltage, resistance, and conductance. All these parameters are complex numbers, represented by real parts, imaginary parts, modules, and arguments.

The concepts needed for this problem domain are:

compl—complex number,
plus —addition of complex numbers,
multi —multiplication of complex numbers,
port —branch of circuit,
cap —capacitor
ind —inductor,
ser —series connection of ports,
par —parallel connection of ports.

The variables needed for this problem domain are:

P —port,
C —complex number,
RE —real part of complex number,
IM —imaginary part of complex number,
MOD—module of complex number,
ARG —argument of complex number,
W —frequency (real number),
C —capacity (real number),
L —inductivity (real number),
I —current (complex number),
U —voltage (complex number),
Z —impedance (complex number),
G —conductance (complex number).

```
%
% Concepts
%
compl(C,RE,IM,MOD,ARG):-
  C=[RE,IM,MOD,ARG],
  (in([RE,IM]),out([MOD,ARG])→cmpl1(RE,IM,MOD,ARG)),
  (out([RE,IM]),in([MOD,ARG])→cmpl2(RE,IM,MOD,ARG)).
plus(C1,C2,C):-
  compl(C1,RE1,IM1,_,_),
  compl(C2,RE2,IM2,_,_),
  compl(C, RE, IM, _,_),
  rel(RE=RE1+RE2),
  rel(IM=IM1+IM2).
multi(C1,C2,C):-
  compl(C1,_,_,MOD1,ARG1),
```

```
    compl(C2,_,_,MOD2,ARG2),
    compl(C, _,_,MOD, ARG ),
    rel(ARG = ARG1 + ART2),
    rel(MOD = MOD1*MOD2)
port(P,I,U,Z,G):—
    P = [I,U,Z,G],
    compl(I,_,_,_,_),
    compl(U,_,_,_,_),
    compl(Z,_,_,_,_),
    compl(G,_,_,_,_),
    multi(I,Z,U),                    % U = I*Z
    multi(Z,G,[1,0,1,0]).            % 1 = Z*G
cap(P,C,W):—
    port(P,_,_,[RE,IM,_,_],_),
    rel(IM = (- 1)/(W*C)),
    rel(RE = IM/1000000).
ind(P,L,W):—
    port(P,_,_,[RE,IM,_,_],_),
    rel(IM = W*L ),
    rel(RE = IM/1000000).
ser(P1,P2,P):—
    port(P1,I,U1,Z1,_),
    port(P2,I,U2,Z2,_),
    port(P,I,U,Z,_),                 % I = I1 = I2
    plus(U1,U2,U),                   % U = U1 + U2
    plus(Z1,Z2,Z).                   % Z = Z1 + Z2
par(P1,P2,P):—
    port(P1,I1,U,_,G1),
    port(P2,I2,U,_,G2),
    port(P,I,U,_,G)                  % U = U1 = U2
    plus(I1,I2,I),                   % I = I1 + I2
    plus(G1,G2,G).                   % G = G1 + G2
%
% Procedures
%
cmpl1(RE,IM,MOD,ARG):—
    MOD is sqrt(RE*RE + IM*IM),
    ARG is atan(IM/RE).
cmpl2(RE,IM,MOD,ARG):—
    RE is MOD*cos(ARG),
    IM is MOD*sin(ARG).
print_port(Port,[I,U,Z,G]):—
    n1,n1,
```

```
  print(Port),nl,
  print('I='),print(I),nl,
  print('U='),print(U),nl,
  print('Z='),print(Z),nl,
  print('G='),print(G),nl.
%
% Example
%
let(W,C1,L2,C3,L4,P1,P2,P3,P4,P5,P6,P):-
  cap(P1,C1,W),
  ind(P2,L2,W),
  cap(P3,C3,W),                %  ---------------------
  ind(P4,L4,W)                 %  |     |     |
  ser(P1,P2,P5),               %  P1    |     |
  par(P5,P3,P6),               %  |     P3    P4
  par(P6,P4,P).                %  P2    |     |
act:-                          %  |     |     |
  W is 6.28*50,                %  ---------------------
  C1=5.0e-9,
  L2=3.0e-3,
  C3=4.0e-9,
  L4=5.0e-3,
  U=[220,0,_,_],
  P3=[_,U,_,_],
  com(let(W,C1,L2,C3,L4,P1,P2,P3,P4,P5,P6,P)),
  print_port('P',P).

|?-act.
P
[[0.000177625,-140.128,140.128,-1.5708],
[220,0,220,0],
[1.99011e-06,1.56999,1.56999,1.5708],
[8.07386e-07,-0.636945,0.636945,-1.5708]]
yes
```

REFERENCES

1. Tyugu, E. (1987). *Knowledge based programming.* Addison-Wesley, New York.
2. Mints, G. and Tyugu, E. (1987). The programming system PRIZ. *Journal of Symbolic Computations* No. 4.
3. McCabe, F. and Clark, K. (1983). *Micro-PROLOG 3.0 Programmer's Reference Manual.* Logic Programming Associates Ltd., p. 135.
4. Clocksin, W. and Mellish, C. (1981). *Programming in Prolog* p. 279, Springer-Verlag, Berlin.

4

On the Construction of Unifying Terms Modulo a Set of Substitutions

S. Langet
Humboldt University,
Berlin, GDR

Abstract

The aim of this paper is to provide a mathematical problem which is of interest for getting practicable methods in the field of inductive program synthesis. The problem we have in mind is obviously a dualism to the unification problem in some equational theory. We consider two terms t_1, t_2 and two possibly different substitutions b_1, b_2 for each of them. The problem is to find a unique term t for both terms such that t and t_1 as well as t and t_2 are unifiable with respect to the underlying equational theory by using b_1 or b_2 as a unifier. The problem is trivially decidable in free-term algebra. It is undecidable in the general case. Moreover, even if the underlying equational theory is decidable, then the problem we have in mind may be undecidable, too. Finally, we derive sufficient preconditions for proving its decidability.

1. INTRODUCTION

Our investigations are closely related to research work in Artificial Intelligence which has been dedicated to inductive program synthesis. The work done in this field has dealt with the problem of automatically synthesizing real software systems from possibly incomplete user-specifications. The incompleteness of some intermediate problem description seems to be a fairly usual occurrence in real programming processes. This reflects, from a theoretical point of view, the gist of inductive inference approaches as described in the survey paper [1], for example.

We will assume that the reader is familiar with the basic ideas of algebraic semantics. For the initial concepts of algebraic semantics he should consult the book by Ehrig and Mahr [3].

The main idea of algebraic semantics is to offer facilities for a stepwise development of software systems on a highly abstract level. Much research has been aimed at producing automatic systems for

†Present address: Leipzig Institute, Department of Mathematics and Informatics, PO Box 66, Leipzig 7030, GDR

synthesizing software systems from algebraic specifications (see [3]). For improving the power of algebraic specification systems, the user should be able to present step-by-step intermediate specifications describing a target software system incompletely. Moreover, the system should be able to learn (or synthesize) automatically a correct specification for the target system (with respect to the underlying semantic concept) in processing incomplete information. For that reason, it has to put together inductive inference algorithms, learning correct specifications from incomplete descriptions presented step by step.

The decidability problem discussed in this paper is of importance if feasible inductive inference algorithms are to be designed for performing this task.

2. BASIC DEFINITIONS AND NOTATIONS

Within the scope of this paper we think of software systems as total algebras of a certain signature. Assume SIG $= (S,O,a)$ to be a finite signature, that is, a finite set S of sort names, a finite set O of names of operators on these sorts, and a mapping a from O to S^+ for fixing the arity of each operator in O. For a collection X containing certain variables for each sort in SIG, $T(\text{SIG},X)$ denotes the set of all well-formed terms over SIG and X. $T(\text{SIG})$ is the subset of all variable-free terms included in $T(\text{SIG},X)$. As usual, finite sets of conditional equations built over SIG and X are considered as specification tools. Assume E to be a finite set of conditional equations. ALG(SIG,E) is defined as the class of all minimal SIG-algebras A, i.e. SIG-algebras finitely generated from their constituents named in SIG such that all conditional equations of E are valid in A. Let us think of ALG(SIG,E) as a category with SIG-homomorphisms from algebras onto algebras as morphisms. $T(\text{SIG})$ is the full term algebra. $T(\text{SIG})|_E$ is the term algebra factorized by the congruence relation induced by E, that is, the initial algebra (up to isomorphism) in the category under consideration.

Our investigations are based on initial algebra semantics. Thus, an algebraic specification, i.e. a pair (SIG,E), describes the algebra $T(\text{SIG})|_E$ (up to isomorphism). It is well known that the initial algebra $T(\text{SIG})|_E$ in ALG(SIG,E) is characterized by the fact that in $T(\text{SIG})|_E$ exactly all term equations are valid which can be proved from E.

In our inductive inference approach objects which should be identified from incomplete descriptions are algebras of a certain signature SIG*. Finite sets E_i' of conditional equations over SIG* and some fixed collection X^* of variables can be presented as incomplete information about some SIG*-algebra A^*. An infinite sequence $E_1', E_2' \ldots$ forms a specification of A^* in the limit (under initial semantics) if and only if the

algebra A^* is initial in the category ALG(SIG*, $\cup E'_n$). The task is to provide algorithms that are able to synthesize large classes of algebras from their specifications in the limit.

Within the scope of this mathematical approach we have studied the following case in detail. Objects to be synthesized are algebras of a certain signature SIG* which can be finitely specified over a given ground specification (SIG,E) (with respect to initial semantics). We consider algebras containing an additional function on the sorts of the algebra specified by (SIG,E). We will assume that the ground specification (SIG,E) is given. Consequently, it will be sufficient to discuss the case that a specification in the limit for an algebra of that kind contains term equations for characterizing the input/output behaviour of the additional function.

In [5] and [6] the author has provided a quite general synthesis methodology for the design of inference algorithms for this case. Algorithms constructed on the basis of this methodology are able to synthesize large classes of algebras which can be finitely specified over different kinds of ground specifications. For the design of practicable algorithms, it seems to be essential that the basic knowledge formalized in a given ground specification (SIG,E) can be considered as a canonical term rewriting system. Using a set E of conditional equations as a term rewriting system, $R(E)$ means 'to read the conclusion of each conditional equation from the left to the right and to use it like a rewrite rule'. (Premises have first to be proved by applying the rules of $R(E)$ itself.) If the corresponding term rewriting system $R(E)$ is canonical, then it defines a decidability procedure about the equational theory induced by E on the set of terms T(SIG) (see [4]). In the sequel, we denote by T_{nf}(SIG) the set of all variable-free terms in T(SIG) which are irreducible. A term t is contained in T_{nf}(SIG) if and only if there is no rule in $R(E)$ which is applicable to t. For a term t in T(SIG) we denote by $nf(t)$ its corresponding and definite normal form in T_{nf}(SIG), that is, t can be reduced to $nf(t)$, and $nf(t)$ is irreducible.

The class of ground specifications we have in mind can be defined as follows.

Definition 1

Let us assume that (SIG,E) is an algebraic specification. Moreover, assume $Z = \langle O_C, O_S, O_T \rangle$ to be a classification of the set of operators O of SIG into three disjoint sets such that $O = O_C \cup O_S \cup O_T$. The triple (SIG,$Z$,$E$) is said to be a classified specification

iff

(1) The corresponding term rewriting system $R(E)$ is canonical;
(2) $\forall \mathrm{op} \in O_S : [a(\mathrm{op}) = s_1 s]$
(3) $\forall t \in T(\mathrm{SIG}) \exists t' \in T(\mathrm{SIG}_C) : [E| = t = t']$

(4) $\forall t \in T(\text{SIG}_C) \exists k \in N \exists t_1, \ldots, t_k \in T(\text{SIG}_C)$
$\forall s \in T(\text{SIG}_S, \{x\})$:
$[E \mid = s[x \leftarrow t] = t_1 \text{ OR} \ldots \text{OR } E \mid = s[x \leftarrow t] = t_k]$

(5) $\forall \text{op} \in O_T : [a(\text{op}) = s_1 \ldots s_k \text{ boolean}]$

Note that '$\mid =$' denotes the usual consequence operator. $T(\text{SIG}_C)$ and $T(\text{SIG}_S, \{x\})$ are the subsets of $T(\text{SIG})$ and $T(\text{SIG}, X)$, respectively, which are formed by restricting the set of operators to O_C and O_S, respectively.

Note that the concepts selector (i.e. the operators in O_S) and constructor (i.e. the operators in O_C) are similar to those defined in [7] which are motivated by the aim to characterize the storage and access behaviour of algebraically specified abstract data types.

3. THE MAIN RESULTS

Assume (SIG,Z,E) to be any classified specification. Moreover, let us assume that SIG* is a signature that differs from SIG in such a way that a new operator 'op' is added to SIG, where the arity of this operator is fixed by $a^*(\text{op}) = s_1 s$. A^* denotes a SIG*-algebra which can be finitely specified over the given ground specification (SIG,E). Additionally, A^* should be characterized by the fact that a target function f_{op} is isomorphically represented by the interpretation of op in A^* (with respect to initial semantics).

In a real program synthesis process based on the general synthesis methodology provided in [5] and [6], the following problem arises: Assume $\text{op}(x_1) = y_1$ and $\text{op}(x_2) = y_2$ to be term equations for representing the input/output behaviour of the function f_{op} in A^*. Thus, x_1, x_2 and y_1, y_2 are variable-free terms in $T(\text{SIG})$ of sort s_1 and s, respectively. In the first step the outputs have to be expressed in terms of the corresponding inputs by means of certain terms in $T(\text{SIG}, \{x\})$. That is to say, terms t_1 and t_2 included in $T(\text{SIG}_C, T(\text{SIG}_S, \{x\}))$ are produced which may be understood as straightforward computations for y_1 from x_1 and y_2 from x_2, respectively. For determining the next steps of the synthesis process, it is essential to know whether or not t_1 and t_2 are instantiations of the same straightforward computation. More formally, one has to decide whether or not there exists a term $t \in T(\text{SIG}_C, T(\text{SIG}_S, \{x\}))$ such that:

$$E \mid = t[x \leftarrow x_1] = y_1 \text{ and } E \mid = t[x \leftarrow x_2] = y_2.$$

In general, this equation yields the following decidability problem.

Definition 2
Assume (SIG,Z,E) to be any classified specification. The problem of the construction of unifying terms is decidable for (SIG,Z,E)
iff

For all terms $t_1, t_2 \in T(\text{SIG}_C, T(\text{SIG}_S, \{x\}))$ and for all substitutions b_1, b_2 it is decidable whether or not there exists a term $t \in T(\text{SIG}_C, T(\text{SIG}_S, \{x\}))$, such that:
$[E \mid = b_1(t) = b_1(t_1)$ and $E \mid = b_2(t) = b_2(t_2)]$.

By reduction to one of the problems well-known to be undecidable, Theorem 1 can be proved. A detailed proof using Hilbert's tenth problem can be found in [6].

Theorem 1
There exists a classified specification (SIG,Z,E) such that the problem of the construction of unifying terms is undecidable for (SIG,Z,E).

For the control of the synthesis process it is of importance to solve the decidability problem defined above. For that reason, we consider the following class of ground specifications.

Definition 3
Assume (SIG,Z,E) to be any classified specification. (SIG,Z,E) is said to be a classified specification with a compatible size-measure $\#$
iff
(1) There is a computable function $\#: T_{nf}(\text{SIG}) \to N$, such that:
(1.1) $\forall t \in T_{nf}(\text{SIG})$: [$\#(t)$ is defined.]
(1.2) $\forall n \in N$: [$\#^{-1}(n)$ is finite.]
(2) $\forall \text{op} \in O_C(a(\text{op}) = s_1 \ldots s_k s) \forall t_1, \ldots, t_k \in T_{nf}(\text{SIG})$:
$[\exists i(i \leq k)$: $E \mid = t_i = \text{op}(t_1, \ldots, t_k)$ or
$\forall i(i \leq k)$: $\#(t_i) \leq \#(nf(\text{op}(t_1, \ldots, t_k)))]$
(3) $\forall \text{op} \in O_S \forall t \in T_{nf}(\text{SIG})$: $[\#(t) \geq \#(nf(\text{op}(t)))]$

Obviously, the function $\#$ defines a size-measure in the sense of Blum [2] over the set of all irreducible terms in $T(\text{SIG})$.

Theorem 2
Let (SIG,Z,E) be any classified specification with a compatible size-measure $\#$. Then it holds that the problem of the construction of unifying terms is decidable for (SIG,Z,E).

A detailed proof of Theorem 2 can be found in [6].

Since the complexity of the decidability problem under consideration determines the complexity of the whole synthesis process, the following result is essential.

Theorem 3
There is a classified specification (SIG,Z,E) with a compatible size-measure $\#$, such that the problem of the construction of unifying terms is NP-complete for (SIG,Z,E).

REFERENCES

1. Angluin, D. and Smith, C. H. (1983). Inductive inference: theory and methods, *Computing Surveys, 15*, No. 3, 237–59.
2. Blum, M. (1967). On the size of machines, *Information and Control*, **11**, No. 3.
3. Ehrig, H. and Mahr, B. (1985). Fundamentals of Algebraic Specifications—Part I, *EATCS—Monographs on TCS*, **6**, Springer-Verlag.
4. Huet, G. and Oppen, D. C. (1980). Equations and rewrite rules: A survey, in: *R. V. Book, Formal Languages: Perspectives and Open Problems*, Academic Press, New York.
5. Lange, S. (1986). A program synthesis algorithm exemplified, in: *Proceedings of MMSSSS '85*, LNCS 215, Springer-Verlag.
6. Lange, S. (1987). A decidability problem of Church-Rosser specifications for program synthesis, in: *Proceedings of AII '86*, LNCS 265, Springer-Verlag.
7. Thomas, M. (1985). The storage and access structure of algebraic specified data types, in: *Abstracts of the 4th Workshop on Specification of Abstract Data Types*, Informatik-Berichte der TU Braunschweig, No. 86–09.

5

Plausible Inference and Negation in Horn Clause Logic

T. B. Niblett

The Turing Institute and University of Strathclyde,
Glasgow, UK

Abstract

We elucidate the role of probabilistic techniques in plausible reasoning and provide reasons why the probabilistic calculus is not adequate by itself for plausible reasoning in expert systems. We suggest a logic-based framework which uses an informal notion of argument as the basis for its representation of certainty. We argue that much of the manipulation of arguments should be done with non-numeric certainties. Probability measures should be introduced only late in the reasoning process, if at all. This scheme is illustrated with examples of the construction and evaluation of arguments. Although much of our work is tentative, we believe that it provides a basis for a well founded knowledge-based approach to plausible reasoning in expert systems.

1. INTRODUCTION

From the beginning of expert systems research, attempts have been made, in systems like MYCIN [1] and PROSPECTOR [6, 5], to devise a calculus of plausible reasoning to deal with uncertain or heuristic knowledge.

There has been considerable dispute about which calculus should be used. Several authors (for example, [2, 5, 27]) have argued for the use of the probability calculus. Others (for example, [3, 1, 22]) find the representational capabilities of the probability calculus inadequate and have argued strongly for alternative approaches. Our view is that the theological intensity of the debate has obscured the fact that the opposed camps are reconcilable. We will attempt to present a view of plausible reasoning that elucidates the reasons for disagreement, and will propose a framework for plausible reasoning that provides a solution to the problems raised.

1.1 The probabilistic model

The probabilistic model of plausible reasoning has two distinct stages.

1. The formation of a model of the problem to be analysed. This is sometimes called the frame of discernment [22]. This model should contain all the relevant information known about the issues at hand.
2. The evaluation, according to the rules of probability calculus of the probabilities of interest, so that a decision can be made.

The problems that confront a mechanization of this process in the expert systems context are twofold. Firstly, evaluation may be difficult because the model produced at stage 1 is under-specified and computationally expensive to evaluate. This problem is discussed in more detail below. Secondly, and perhaps more fundamentally, there are considerable problems associated with the model and the process of model formation.

1. The language of probability statements over propositional variables, which is the 'standard', is an impoverished representation language. It is difficult to mesh with first order logic, which is more desirable as a general-purpose representation language.
2. Updating beliefs proceeds by conditionalization. It is more natural in many cases to modify the model on receipt of new evidence. It is not clear how this should be done within the probabilistic framework, where the only mechanism available is conditionalization.
3. There are reasons to believe that, in many cases, people regard the comparison of certainties as being impossible or at least undesirable. Polya [20] gives as an example the question

   'Is it more probable that the Vikings discovered America or that Goldbach's conjecture is true?'

1.2. Alternative approaches and synthesis

There has been considerable discussion about alternatives to the probabilistic approach. Proponents of the probabilistic view argue quite persuasively that, when decisions are to be made, the probabilistic approach is the only one which meets justifiable criteria for rational decision-making. These criteria are:

1. Any two probabilities are comparable. This implies a total ordering on probabilties, and with the next two assumptions assigns all probabilities to the interval [0, 1].
2. Every event has probability ≥ 0.
3. The certain event has probability 1.
4. The probability of the disjunction of two disjoint events is equal to the sum of their individual probabilities.

5. The belief in p and q given evidence e ($B(pq|e)$) is a continuous monotone function of $B(q|e)$ and $B(p|qe)$.

We concede that these arguments offer a compelling reason for the use of probabilities. This does not, however, mean that there is a straightforward way in which probabilities can be used universally within knowledge-based systems. The assumption that certainties can be assigned numeric values has been argued against above. These arguments are symptomatic of two broad issues, which stem from the two components of the probabalistic model discussed above. These are, *knowledge representation* in the sense of building models to which probability arguments can be applied and *computational constraints* which relate to the complexity of straightforward implementations of probability models. We tackle these issues separately.

1.3. Knowledge representation issues

It is a truism to state that the power of knowledge-based systems resides in the knowledge. Two basic principles in the development of such systems are that knowledge be represented explicitly wherever possible and that the representation scheme used, together with its inference mechanism should be able to represent and reason about all the necessary knowledge. We feel that much of the confusion in the debate about the relative merits of probability theory as a formalism has resulted from concentration on the second of these principles, rather than the first. The vigorous defence of probability in [2] argues persuasively that standard Bayesian theory can adequately represent the necessary kinds of uncertainty, if the concept of higher order probability is included. At the same time Cheeseman admits that a conditional probability for an event or proposition is a *summarization* of the total evidence for the event or proposition in question. As it stands this violates the first principle above.

A similar difficulty occurs with the vexed problem of *priors*. A Bayesian asserts that the subjective prior distribution is important because it provides a systematic method by which a large amount of heterogeneous information can be summarized. Again, in the knowledge-based context the objection is that this information is not accessible to the reasoning agent.

It is of fundamental importance to realize that in the context of scientific discovery or experimentation the particular form of the prior is not crucial. With sufficient observational data the conclusion (or posterior distribution) is indifferent to the choice of prior. This is clearly not the case with most knowledge-based systems where a single 'experiment' is usually conducted. In this case the prior distribution has a significant

effect on the final decision. Thus, more importance attaches to the system's ability to justify the prior.

Let us review the problems discussed above with respect to representation and consider them in more detail. The problems were:

1. the limitations of the conventional propositional language;
2. the problem of updating beliefs by conditionalization;
3. inappropriate comparison of certainties.

1.3.1. *Language limitations*

Variants of propositional languages have been used almost exclusively by Artificial Intelligence (AI) workers involved in plausible reasoning. Little work has been done on first order probabilistic logics. Much of the necessary theoretical background has been provided by [4] and [23]. We will use the Horn clause variant of first order logic, which provides a more powerful representation language. The certainty calculus is not directly related to first order probabilistic logic.

1.3.2. *Updating beliefs*

The sole mechanism for belief updating provided in the probabilistic calculus is conditionalization by use of Bayes' theorem. There are various related objections to this.

It is often more natural to change the frame of discernment when new information impinges, rather than to move to a new probability distribution by conditionalization.

It has been argued [2] that the complex relationships between propositions that are handled by truth maintenance systems are subsumed by the appropriate specification of probability distribution. In the terminology of [13] the theory is epistemologically adequate. There remains, however, the problem of providing a computational mechanism for such updating. This mechanism is not provided within probability theory.

There are further representational problems with the exclusive use of conditionalization, such as the problem of asserting that an event has in fact occurred [28], which are outside the scope of this paper.

In the language of machine learning you want to shift the bias. Any objection by Bayesians on the grounds of the principal of total information ignores the computational issues discussed below.

1.3.3. *Comparison of certainties*

Probabilities can always be compared. Sometimes this is not appropriate. There should be a mechanism for deciding when comparisons can

be made. Note: this works for 'pre-compiled' networks like PROSPECTOR where comparison is guaranteed to work by the construction of the network.

1.4. Complexity issues

Consider the propositional network of Figure 1. There are n concepts $(E_1 \ldots E_n)$ and $o(n)$ rules, represented by links between concepts—a rule *if* E_j *then* E_k is shown by a link between E_j and E_k. The value of a node without any descendants can be supplied by the user or supplied as an *a priori* value. The problem that now confronts this system, whatever calculus of plausible inference it uses is that the rules, together with the inferential calculus will only determine $o(n)$ constraints on the probability distribution of the E_i. To specify the distribution completely, however, we need 2^n constraints. This means either that some sort of blanket assumption must be applied to determine the distribution, or that the upper and lower bounds of relevant probabilities must be maintained. In practice it is found that maintaining bounds without assumptions is not fruitful as the bounds are 0 and 1 (or nearly so). Any assumptions that are made tend to be unjustifiable except in special cases (see, for example, [9, 19]).

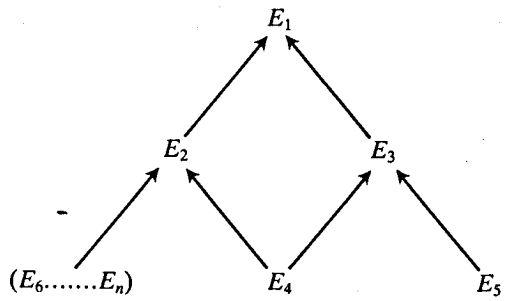


Figure 1. A simple model of an inference system.

It is worth singling out one method for making such assumptions; the method of least information [7, 10, 2]. The rationale of this method is to specify as much knowledge as possible about a distribution in terms of independence, marginal distributions, and so on, then to choose the distribution that adds as little information as possible. Good [7] shows that this is equivalent to assuming independence of higher order interactions between events. The objections to this method as a uniform mechanism are twofold.

1. There are computational problems—in general the problem is exponential in the number of propositions.
2. The method is very sensitive to the model used and the constraints

provided. A somewhat misleading analogy with statistical mechanics (for example, [10]) is sometimes used to justify this procedure. Our objection is, as before, that without a mechanism for examining the assumptions that have been made and being able to modify the underlying model this technique is not generally applicable.

A further criticism that can be levelled at the probabilistic approach on computational grounds is that the formalism does not lend itself to judging when information input by the user is inconsistent. It has been argued in [2] that techniques of default reasoning and the use of ATMS systems is subsumed by the probabilistic approach. Again we must emphasize that this is not the case where interactions are complex and track must be kept of assumptions on an individual basis. This can only be done independently of the probability calculus. Cohen [3] argues this in his case for non-numeric certainties.

1.4.1. *Summary of objections*

The following is a summary of our objections to the use of the probability calculus as the *sole* mechanism for belief updating.

1. It provides no assistance in the construction of a model of the problem. As anyone who has done statistical analysis of data is aware it is dangerous to apply general solutions to particular problems in a rote fashion.
2. Conditionalization is not an adequate mechanism by itself to accommodate a computational description of belief revision.
3. Computational limitations require assumptions to be made about distributions which cannot be represented within the probability calculus.

We must emphasize that these objections are not to the use of Bayesian techniques *per se*, but to their use as the sole knowledge representation formalism in a knowledge-based system.

1.4.2. *An alternative methodology*

The above analysis suggests requirements for a computational framework. The central notion is of an *argument* for a statement or proposition. We argue that in expert systems the primitive notion be that of an *argument* for a concept's plausibility, and that at the level of knowledge representation judgements of truth or probability are secondary. From this viewpoint the traditional role of logic is to provide rules for the construction of *valid* arguments, and valid arguments have in common the property of establishing the *truth* of their entailed propositions. The

difference from the traditional notion of validity is that we are concerned with the internal structure of proofs, as well as with their existence or non-existence. Different arguments for the same proposition may have different degrees of validity. From this point of view the role of probabilistic reasoning in expert systems is to provide numerical values for arguments or their components for the purposes of comparison. That these numbers are interpreted as probabilities ensures coherent results.

A rough equivalence with purely probabilistic systems can be drawn up as in Table 1.

Table 1. Argument-based vs. probabilistic system.

Consistent assignment of priors	Rules defining the circumstances in which assumptions can be introduced
Detecting contradictions	Maintaining the consistency of arguments with constraints
Weighing evidence	Constructing and comparing arguments

The framework we use is Horn clause logic. This provides a concrete basis with which to illustrate the usefulness of our ideas, and has a straightforward semantics. Technically our approach is to extend the domain of truth values to include non-numeric certainties. Several questions are raised about the notion of argument which we discuss below.

Question 1 *What is an argument?*

Our approach is pragmatic, in that we adopt the existing framework of Horn clause logic and its implementation via PROLOG.

We consider an argument for an atom to be a *proof tree* for that atom. A proof tree is an 'exploded' version of the SLD proof for an atom. Figure 2 illustrates the proof tree for the goal *?- append([1],[2],[1,2])* (in PROLOG syntax).

A goal can have more than one proof tree. As far as two-valued logic goes this is immaterial, any proof of a (ground) atom will do. From our perspective two different proofs of the same ground atom will have different plausibilities. In our model therefore the truth value of an atom is a function of all the proofs for that atom. We shall see that modelling of a PROSPECTOR-style inference network requires that certainties are calculated from all proofs rather than just one.

Question 2 *How do you construct, compare, and combine arguments that are* (a) *mutually supporting, or* (b) *mutually contradictory?*

The answer to this question is critical to our approach, and is largely determined by the type of certainty that we allow. As discussed above it

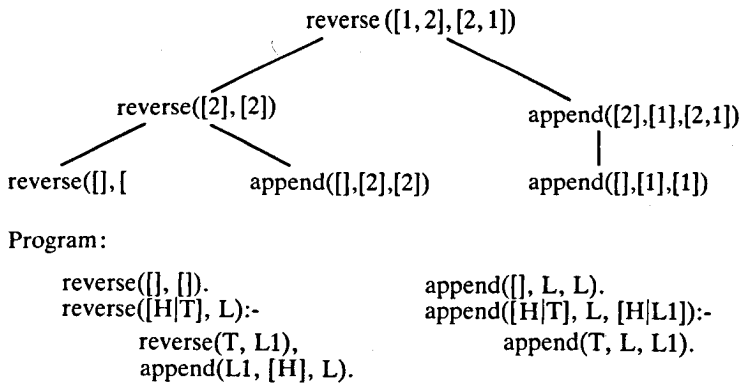


Figure 2. A proof tree for the goal reverse ([1,2], [2,1]).

is often undesirable that we be able to compare certainties directly. Furthermore, we would like a framework for certainties that allows symbolic reasoning to be performed. If two certainties are incomparable, for example, we may wish to reconcile this with some form of rule-based reasoning.

The natural structure for such certainties is a lattice. This is a partially ordered set with operations of meet ($\sqcap$) and join ($\sqcup$). These operations correspond to logical 'and' and 'or' respectively, and together with the partial order form the basis of comparison and combination. In general, if two certainties are not comparable these operations may not be defined. Examples of such lattices are given in Section 3.1 and Section 3.2.

An interesting question is posed by the problem of negation when we use the Horn clause approach. Previous work [17, 26] did not treat negation. Our proposed system handles negation as a modified form of negation as failure. The problems of providing a semantics for this form of negation are similar to those of Horn clause logic, and can be handled in much the same way. The use of some form of negation is necessary to provide the expressive power of, for example, PROSPECTOR. When numerical certainties are used this reduces to the probabilistic case where $B(\bar{p}) = 1 - B(p)$.

It is not clear that this approach is adequate to capture common uses of negation in arguments.

Question 3 *What does it mean for an argument to be consistent (coherent)?*

Arguments for or against propositions are central to our approach. It is important that arguments are coherent. There are several kinds of coherence.

(1) consistent in the logical sense;

(2) admissible in the legal sense;

(3) comprehensible and relevant to a user;

(4) plausible to some degree.

All of these interpretations are useful in certain contexts. The desirability of a computational approach other than the probability calculus was mentioned above in the context of consistency. The question of the admissibility of arguments, which arises very commonly in legal reasoning, does not seem directly amenable to a probabilistic approach either.

Our approach is to use the meta-logical capabilities of PROLOG and the certainty space to represent these different notions. We illustrate this by reference to the consistency issue.

Truth maintenance systems, as discussed in [11], use a propositional representation for assumptions and detect inconsistency using a suitable form of inference (decidable for propositional logic). In Section 3.2 we show that a certainty lattice can be used to disallow inconsistent arguments by assigning truth value *false* to them. In particular the argument itself can be expressed in first order logic while the certainties are propositional. Thus, the arguments can be phrased in a more expressive language and the certainties in a decidable sublanguage.

We do not address the other issues relating to coherence in this paper.

1.5. Related approaches

Shapiro [26] describes a method of extending Horn clause logic to handle uncertainty. The emphasis of his work differs in that he is not concerned with the use of arguments or non-numeric certainties. In addition he does not discuss the problem of negation. Cohen [3] describes a system of non-numeric certainties or endorsements that can be used to reason about the uncertainty of arguments for and against hypotheses. Cohen does not provide a formal definition of argument or of the methods to be used for reasoning about the uncertainties used. The system presented here seems to use a notion of argument, based on the proof procedure of PROLOG programs (SLD resolution), which is more limited than Cohen but attempts a more formal definition of the notions used.

In the next section we shall briefly describe the framework of our certainty calculus (described in more detail in [17]) and describe its extension to negated goals.

2. THE FORMAL APPARATUS

2.1. Basic definitions

Definition A *certainty space C* is a complete lattice. That is a partially

ordered (under $\leqslant$) set with operations $\sqcup$ and $\sqcap$ (least upper bound and greatest lower bound respectively) defined for every subset of C. The greatest and least members of C are $\top$ and $\bot$ respectively (sometimes called *true* and *false*).

Given a certainty space C we define C^+ to be the smallest complemented distributive lattice with C as a sublattice.

The lattice C provides the framework for certainty values in generalized Horn clause programs. The extended lattice C^+ provides certainties for general programs with negative literals in the bodies of clauses.

Definition We extend $\leqslant$ to sequences over C by defining: $\langle c_1, \ldots, c_n \rangle \leqslant \langle c_1', \ldots, c_n' \rangle$ iff $c_1 \leqslant c_1'$ and ... and $c_n \leqslant c_n'$.

Definition A function f from sequences of certainties to certainties in some certainty space C is *monotone* iff for all sequences s_1 and s_2 of length n over C $s_1 \leqslant s_2$ implies $f(s_1) \leqslant f(s_2)$.

Definition A *general clause* has the form $A \leftarrow B_1, \ldots, B_k$ where A is a positive literal and the B_i are positive or negative literals.

Definition A *general logic program with uncertainties* is a finite (non-empty) set P of pairs of the form $\{\langle A \leftarrow B, f \rangle\}$ where $\{A \leftarrow B\}$ is a general clause, and f is a monotone function from sequences of certainties to certainties. We shall sometimes write $f_{A \leftarrow B}$ for the function f paired with clause $A \leftarrow B$.

It should be noted that the defintion of certainty function is more general than that provided by Shapiro [26] in that it recognizes the order of goals within a clause.

2.2. Semantics

We can now define the semantics of general logic programs with uncertainties as an extension of that given by [17] for Horn clause programs with uncertainty.

Definition The Herbrand Universe *U(P)* is defined recursively as

(1) The set of constant symbols in P (or the constant symbol a if there are none);

(2) All atoms of the form $p(t_1, \ldots, t_n)$ where the t_i are in *U(P)*.

Definition The Hebrand base *H(P)* of a logic program P is the set of all ground atoms formed by using predicate symbols from P with ground terms from the Herbrand Universe *U(P)*. The set of all ground instances of an atom A is written $GI(A)$.

Definition An *interpretation* I of a general logic program with uncertainties P is a function from *H(P)* to a certainty space C.

An interpretation I can be extended in the natural way to an interpretation I^+ by defining $I(\neg P) = I(\bar{P})$ (where the complement of $c \in C^+$ is written $\bar{c}$). We shall identify I with this extended interpretation in the following text.

Definition An interpretation I_1 is $\leqslant$ an interpretation I_2 iff $I_1(a) \leqslant I_2(a)$ for all a in $H(P)$.

Definition A model M of P is an interpretation of P satisfying the following conditions:

For any pair $\langle A \leftarrow B, f \rangle$ in P and any ground instance $A' \leftarrow B'_1 \ \& \ldots \& \ B'_n$ of the clause, then $M(A') > = f(M(B'_1), \ \ldots, \ M(B'_n))$, where $M(\neg L) = M(\bar{L})$ as above.

2.2.1. *Model theoretic semantics*

With Horn clauses instead of clauses with negated literals the model intersection property holds, where the pointwise intersections of two models is itself a model. This does not hold for general programs, as can be seen from the following example. Let $P = \{\langle a \leftarrow \neg b, f \rangle, \langle b \leftarrow \neg a, f \rangle\}$ with certainty space $\top, \bot$ and $f(c) = c$ for $c \in \{\top, \bot\}$. There are models with just a true or just b true, but the intersection with neither true is not a model.

Despite this we can define a *minimal model* M^{min} for a program P as a model having the property that if $M' \leqslant M^{min}$, M' a model, then $M' = M^{min}$. Minimal models for a program P exist (since for any P the interpretation $I(a) = \top$ is a model and by use of Zorn's lemma). Unfortunately, as the above example shows, there is no globally minimum model as there is for the Horn clause case. We are now in a position to define the certainty of a ground atom A with respect to a program P (where $A \in H(P)$).

Definition The certainty $c(A)$ of a ground atom $A \in H(P) = \prod_{M a model} M(A)$.

Definition The certainty of an atom A (i.e. $\exists A$) with respect to program P is $\bigsqcup_{a \in GI(A)} c(a)$.

When the certainty space C is the 2-element lattice, this definition reduces to the usual definition of truth as 'true in all models'.

In [17] it is shown that if a program consists of pure Horn clauses then a constructive fixpoint definition of the minimal model for the program can be given.

2.2.2. *Inference procedure*

In this section we shall define an inference procedure SLDC resolution (the 'c' for certainty), based on SLD resolution for logic with certainties

and we shall show that this procedure is correct in the sense that when it terminates, the estimate of certainty produced is the model-theoretic certainty of the (ground) atom (for a definition of 'SLD' resolution and other terms see [12]).

Definition Let P be a program, G a goal and R a safe computation rule. An SLDC tree for $P \sqcup \{\leftarrow G\}$ via R is defined as the smallest tree satisfying the following criteria:

1. Each node of the tree is a conjunct of literal/certainty pairs.
2. The root of the tree is $\leftarrow G/C_G$ where C_G is the certainty of G.
3. For each node $\leftarrow p_1, \ldots, p_n$ $(n \geqslant 1)$ in the tree, if R selects for a positive literal/certainty pair $p_i (\equiv l_i/c_i)$ then the node has a descendant for each clause/function pair $\langle H \leftarrow B_1, \ldots, B_k, f \rangle$ in P such that l_i and H are unifiable. The descendant node is

$$\leftarrow (p_1, \ldots, p_{i-1}, b_1, \ldots, b_k, p_{i+1}, \ldots, p_k)\theta$$

where $b_j = B_j/c_{b_j}$, $j \in [1,k]$ and where θ is a most general unifier of l_i and H. The certainty c_i is equal to

$$\bigsqcup_{\text{descendant nodes}} f(\langle c_{b_1}, \ldots, c_{b_k} \rangle)$$

4. For each node in the tree if the selected pair $p_i = \neg l_i/\bar{c}_i$ is negative (and ground) then the root of the SLDC tree for l_i *is* $\leftarrow l_i/\bar{c}_i$ and the single descendant of the node is $\leftarrow p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$.
5. Nodes which are empty have no descendants.
6. Nodes $\leftarrow l_1/c_1, \ldots, l_n/c_n$ which are non-empty and have no descendants have $c_i = \bot$ for all i.

For any given atom all SLDC trees (for a given computation rule) are identical modulo a renaming of variables. Hence we speak of 'the' SLDC tree for an atom. The SLDC tree as defined here will form the basis of a computational object. To be of real use we require that the tree be computable in finite time, hence the following definitions.

Definition The *index* of an SLDC tree is 0 if the tree contains no negative subgoals, otherwise it is equal to the number of negative subgoals plus the sum of the indices of the SLDC trees of its negative subgoals. An SLDC tree is *finite*, if it has a finite number of nodes and there is an integer N such that the index of the tree is less than N.

With this definition of SLDC tree we can now show that if a ground atom A has a (finite) SLDC tree with root $\leftarrow A/A_c$ then A_c is equal to the model-theoretic certainty of A.

Theorem If A is a ground atom and has a finite SLDC tree T with root node $\leftarrow A/A_c$ then A_c is the certainty of A.

Proof We proceed by induction on the index of the SLDC tree.

Index = 0 We proceed by induction on the depth of the tree. If the depth is 1 then from the definition of SLDC tree we have root node $\leftarrow A/C_A$ where

$$C_A = \bigsqcup_{H\leftarrow} f_{H\leftarrow}(true)$$

here H and A have a most general unifier. Let C be the certainty of A then by the definition of satisfaction for clauses $C \geqslant f_{H\leftarrow}$ for each H. Thus $C \geqslant \sqcup_{H\leftarrow} f_{H\leftarrow}(true)$. To show that equality holds we note that the above lub is the lub of certainties over all ground instances of clauses for A. From the definition of C the equality must hold.

If the depth of the tree is k (>1) then consider the descendants of the root node $\leftarrow A/C_A$. Each such node is isomorphic to a conjunction of SLDC trees. Applying the inductive hypothesis each goal (as the root of an SLDC tree of depth $< k$) has as associated certainty its true certainty. Applying a similar argument to that of the previous case, the hypothesis is proved.

Index = $k > 0$ Applying the inductive hypothesis to each negative literal, we see that its SLDC tree returns the correct certainty. The SLDC tree for A must therefore return the correct certainty.

2.3. Discussion

We have demonstrated the soundness of SLDC resolution. Whenever an SLDC tree is finite the value assigned to its root is the certainty of the atom at the root. It is straightforward to construct an interpreter in PROLOG to find such trees.

The interpreter will fail to terminate whenever the corresponding SLDC search tree is infinite. In practice this is not inconvenient. If approximations to the truth values of literals are sufficient then it is possible to build interpreters that will terminate on a larger set of root literals. For such approximations to be *underestimates* of the true values we still require that the SLD trees corresponding to negated goals are finite.

The SLDC tree corresponds to all possible arguments for an hypothesis. A single argument corresponds to a branch of the SLDC tree, as a proof for a PROLOG goal corresponds to a single branch of its SLD tree.

In some circumstances (see Section 3.2) the SLDC tree is decomposable in the sense that the certainty of the root goal can be derived from the certainty of the individual arguments for the goal. This property is important for expert systems, where it is more natural, in many cases, to work with individual arguments.

In the next two sections we shall see two different implementations of this calculus, both of which use negation.

3. IMPLEMENTATION EXAMPLES

We shall describe two implementations of specific inference mechanisms within our framework. The two implementations differ in the certainty lattice used. Both implementations use the abstract interpreters of Figure 3. The first interpreter constructs single arguments for or against a proposition, the second constructs all arguments possible from the input program. Both interpreters use the default top-down inference strategy of PROLOG. The difference is that the first evaluates a single argument, while the second evaluates all arguments. Goals that involve lattice calculations are italicized.

```
% Interpreter for single arguments

solve((A,B), Certainty):—
   solve(A, CertA),
   solve(B, CertB),
   combine_and(CertA, CertB, Certainty).

solve(not A, Certainty):—
   solve(A, Min, NotA),
   lattice_not(NotA, Certainty).

solve(Goal, Certainty):—
   clause(Function, Goal, Body),
   solve(Body, BodyCert),
   lattice_func(Function, BodyCert, Certainty)

% Interpreter to evaluate all arguments

solveall((A,B), Certainty):—
   solveall(A, CertA),
   solveall(B, CertB),
   lattice_and(CertA, CertB, Certainty).

solveall(not A, Certainty):—
   solveall(A, Min, NotA),
   lattice_not(NotA, Certainty).

solveall(Goal, Certainty):—
   findall( Cert, (
      clause(Function, Goal, Body),
      solveall(Body, BodyCert),
      lattice_func(Function, BodyCert, Cert)),
      CertList),
   lattice_or(CertList, Certainty).
```

Figure 3. Two interpreters in PROLOG.

3.1. Example: A PROSPECTOR-like scheme

To demonstrate the range of our framework, we implement the plausible reasoning mechanism used by the PROSPECTOR expert system, and show that this can be expressed within our framework. As we wish to show that certainties for arguments can be combined we shall use the *weight of evidence* for a proposition as the measure of certainty.

Following Good [8] we define the weight of evidence for concerning H provided by E ($W(H{:}E)$) as:

$$W(H{:}E) = \log(P(E|H)/P(E|\bar{H})) \tag{1}$$

$$= \log(O(H|E)/O(H)) \tag{2}$$

The weight of evidence has the nice property that if E_1 and E_2 are conditionally independent given both H and $\bar{H}$ then $W(H{:}E_1E_2) = W(H{:}E_1) + W(H{:}E_2)$.

We shall use propositional (variable-free) rules throughout. The PROSPECTOR formalism is itself based on a propositional description of events. Our presentation focuses on showing that the probabilistic inference mechanism of PROSPECTOR can be re-phrased in terms of weights of evidence. There are three types of combination that need to be considered:

(1) the evaluation of *if-then* rules (including chaining);

(2) the evaluation of Boolean combinations of events;

(3) the combination of multiple rules for an event.

3.1.1. *Evaluation of* if-then *rules*

We have a rule of the form *if E then (probably) H.* In general an observation E' is made that provides an estimate of the probability of E. We can calculate as follows.

$$p(H|E') = p(HE|E') + p(H\bar{E}|E') \tag{3}$$

$$= p(H|EE')p(E|E') + p(H|\bar{E}E')p(\bar{E}|E') \tag{4}$$

$$\text{(Markov)} \quad = p(H|E)p(E|E') + p(H|\bar{E})p(\bar{E}|E') \tag{5}$$

This calculation involves a Markov-type assumption that if the probability of E is known with certainty (true or false) then the observation E' does not affect the conditional probability of H.

We can now calculate the posterior odds of H given E'.

$$O(H|E') = \frac{p(H|E)p(E|E') + p(H|\bar{E})p(\bar{E}|E')}{p(\bar{H}|E)p(E|E') + p(\bar{H}|\bar{E})p(\bar{E}|E')}$$

Recall that $W(H{:}E') = \log(O(H|E')) - \log(O(H))$. To calculate this

weight we require an estimate of the prior odds on H. In addition, if we wish to phrase the information provided by E' in terms of a weight of evidence for E we require a prior for E†.

We shall now show how updating can be performed using weights of evidence. For convenience we define the following parameters.

$$\alpha = p(H|E) \quad \beta = p(H|\bar{E}) \quad \eta = P(H) \quad \gamma = p(E)$$

Let $\omega = W(E:E')$. Then we can calculate $p(E|E') = \lambda'$ in terms of the weight of evidence. $P(E|E') = \gamma' = \gamma 2^{\omega}/(1 + \gamma(2^{\omega} - 1))$. Similarly

$$W(H:E') = \log \frac{\alpha\gamma' + \beta(1-\gamma')}{(1-\alpha)\gamma' + (1-\beta)(1-\gamma')} - \log \frac{\eta}{1-\eta}$$

This demonstrates an updating mechanism that works solely in terms of weights of evidence. A particular rule is determined by the four parameters α, β, η, and γ. These parameters are determined by the expert. Events E that are supplied by the user can be determined either from γ' or directly from ω.

3.1.2. *Boolean combinations of events*

The rules used by PROSPECTOR for Boolean combinations of events are based on fuzzy logic. In terms of weights of evidence they are expressed as follows:

$$W(H_1 H_2 | E) = \min\{W(H_1|E), W(H_2|E)\} \tag{6}$$

$$W(\bar{H}|E) = -W(H|E) \tag{7}$$

$$W(H_1 \text{ or } H_2|E) = \max\{W(H_1|E), W(H_2|E)\} \tag{8}$$

3.1.3. *Combining multiple evidence*

Given two rules providing evidence for H (*if* E_1 *then* H and *if* E_2 *then* H) we can sum the weights of evidence, provided that we make the assumption which is made in PROSPECTOR that E_1 and E_2 are conditionally independent given both H and $\bar{H}$.

3.1.4. *Implementation of the* PROSPECTOR *mechanism*

It is straightforward to verify that the updating function of Equation (1) satisfies the monotonicity condition, as do the Boolean operations of Equation (2). We shall now provide PROLOG versions of the lattice relations of the above interpreters.

†This can lead to an inconsistent formulation. In PROSPECTOR this was accepted by performing a piecewise linear approximation to the above equation.

```
lattice_and (WeightA, WeightB, Weight):—
        max(WeightA, WeightB, Weight).

lattice_or ([],0),
        lattice_or([Weight − Weights], Sum):—
        lattice_or(Weights, TailSum),
        Sum is Weight + TailSum.

lattice_not(Weight, NegWeight):—
        NegWeight is 0-Weight.

lattice_func(func(α,β,γ,η), Weight, Result):—
        γ' is γ2^Weight/(1 + γ(2^Weight − 1),
```

$$\text{Result is } \log \frac{\alpha\gamma' + \beta(1-\gamma')}{(1-\alpha)\gamma' + (1-\beta)(1-\gamma')} - \log \frac{\eta}{1-\eta}.$$

The form of Equation (1) shows us that the updating function is not additive in the sense that the sum of weights for each argument for an hypothesis H does not necessarily provide the correct weight for H as is calculated by the predicate *solveall.*

3.2. Non-numerical certainties

We now describe a non-numerical system of certainties, illustrative of our approach to plausible reasoning. This system is a simplified version of an implementation in the YAPES expert system shell [18]. The aims of this system are:

(1) to make explicit the use of arguments;

(2) to facilitate the use of meta-level reasoning about uncertainty;

(3) to demonstrate how user interaction can be implemented in conjunction with uncertain reasoning.

3.2.1. *The certainty lattice*

The certainty lattice we consider has as its basic elements ground literals, each literal having an associated justification (written as literal/justification). The justifications are included solely to provide information to the user of the system. These literals stand for assumptions which, if true, ensure the truth of their associated goals (with respect to a given program). The basic certainties are treated as propositions in that their internal structure as terms is ignored. Thus the literal sex(male) is unconnected with the literal sex(female). The lattice operations of $\sqcup$, $\sqcap$ and $\neg$ are implemented as the propositional operations or, and, and $\neg$ respectively. The lattice has a bottom element *false* and a top element *true.* This lattice satisfies the requirements of Section 2. The implementation

of the lattice operations uses a propositional simplifier based on Wang's algorithm [21] which reduces expressions to disjunctive normal form.

The reason for using this lattice in the expert system context is to minimize the number of questions that the user is asked, without compromising the accuracy of results.

A simple example taken from the domain of claiming travelling expenses is used in illustration. One question that we may need to ask is whether the trip was at home (within the United Kingdom) or abroad. We may wish to specify that it should be assumed that the trip is within the UK if the distance travelled on the trip is known to be less than 300 kilometres. The clause for this is written as follows:

```
assume::location(home):—
        distance(K)
        K < 300.
```

where *::* is defined as an infix operator. In general a clause of the form *assume::H:—B* corresponds in our certainty calculus to the clause with certainties $\langle H \leftarrow B, f \rangle$ where $f(C) = C$ and H/B.

If we assume that the distance travelled was 200 km then the certainty of the literal *location*(*home*) is *location*(*home*)/(*distance*(200),200 < 300).

3.2.2. *Calculating certainties*

We are now in a position to consider the evaluation of certainties. The first interpreter of Fig. 3 can be extended with the following lattice operations:

```
lattice_and(A, B, AandB):—
  simp(A and B, AandB).
lattice_or(A, B, AorB):—
  simp(A or B, AorB).
lattice_not(A, ANot):—
  simp(¬A, ANot).
lattice_func(Just, Body, Head):—
  simp(Just and Body, Head).
```

This simple interpreter returns values that do not have a straightforward connection with the model-theoretic certainty. The reason for this is its behaviour on negated goals. If an argument contains no negated goals then the certainty returned will be an underestimate of the model-theoretic certainty. The value returned for negated goals is, however, an over-estimate of the model-theoretic certainty. The second interpreter of Fig. 3 will return the correct certainty. We should note that the propositional lattice is additive in the sense that the disjunction of the

certainties of all the separate arguments for a goal H is equal to the certainty returned by the second interpreter.

In practice it is not clear that this more 'correct' treatment of negation is either necessary or desirable. In many cases of practical reasoning this implicit use of the closed world assumption is not considered to produce valid arguments (in a law court for example).

Let us consider the problem of deciding the location of an employee's trip in a little more detail. The clause we considered specifies conditions under which it is justifiable to assume the location was 'home'. As formulated this provides no information as to the goal *location(abroad)*. The interpreter of Fig. 3 will fail on this goal, returning the value *false* by negation as failure. This is unsatisfactory for two reasons.

1. Typically, when creating a knowledge base the programmer will have in mind the fact that a trip is either at home or abroad. With this additional knowledge our original clause is relevant to the goal *location(abroad)*. Broadly speaking if the distance travelled was 200 km we are entitled to say that the trip was not abroad because we assume that it is at home, given the distance actually travelled.
2. Horn clause logic cannot express the fact that the locations 'home' and 'abroad' are mutually exclusive.

We deal with (2) by operating at the meta-level with the propositional simplifier. We add a meta-level assertion to the propositional simplifier for certainties that the conjunct of *location(X)* and *location(Y)* with $X \neq Y$ reduces to *false*. This assertion is used only if the variables are instantiated. Use of such constraints within the simplifier ensures that a particular argument will be consistent.

We deal with (2) at the meta-level as well. In this case we assert that the goals *location(abroad)* and *location(home)* can be generalized to the goal *location(Place)* which has one and only one solution. The implementation involves the addition of a new clause to the interpreter of Fig. 3. The following clause is added before the third clause for *solve*.

```
solve(Goal, Certainty):—
  generalizes(Goal, General), !,
  solve(Goal, PlusValue),
  findall(Val,(solve(General, Val),
    General ≠ Goal), MinusList),
  lattice_or(MinusList, Minus),
  lattice_not(Minus, Plus2Value),
  lattice_and(PlusValue, Plus2Value, Certainty).
```

The goal *generalizes(Goal, General)* means that the generalization *General* of *Goal* has one and only one solution. The issue of maintaining

consistency arises in the above example, where it is inconsistent to assume that the trip is both at home and abroad. It is not possible to make both assumptions simultaneously because the conjunct of these two certainty values is defined to be false. In this case either one assumption or the other may be made, but not both. The detection of such inconsistencies is effective because we are dealing with the propositional calculus, which is decidable. This aspect of our implementation has some similarities with the assumption-based truth maintenance system of De Kleer [11].

3.2.3. *User interface*

An important property of expert systems is that they require considerable interaction with the user. Following Sergot [24] we can implement user interaction by considering the user as a source of unit clauses. This distinction should be transparent to the inference mechanism of the expert system.

Let us consider our example concerning the location of an employee's business trip. We assume that the user of the system knows the location, the inference system is making an assumption to avoid having to ask the question of the user. It is not always the case that assumptions are made about relations that can be asked of the user. It may be that there is genuine uncertainty involved, as in the domain of geological interpretation modelled by PROSPECTOR. It may also be the case that the user may not know, in some circumstances, the truth-value of a relation.

The strategy pursued in YAPES and described here is to ask the user about a relation only when it would otherwise be proven false (by negation as failure). The goal *location(abroad)* in the above example would therefore be assumed false since we assume that the location is *home*. This strategy can be readily implemented when we are looking for a single argument for each goal. It is sufficient to add the following clause as the last clause of the *solve/3* relation.

```
solve(Goal, true):—
  askable(Goal),
  ask_user(Goal).
```

When we require all arguments for a goal we must define two modes of search: the first where the user is not asked, and the second where the user is asked. Again this distinction must be made at the meta-level.

4. CONCLUSIONS

We have demonstrated a system of plausible inference, based on Horn

clause logic, which differs from other systems of plausible inference in its emphasis on arguments. We have illustrated the scope of the system by implementing a PROSPECTOR-like plausible inference system as well as a non-numerical system. The advantages of the system can be summarized as:

1. A formal semantics which guarantees the correctness of results. In particular we can say with precision that when all the negative goals are evaluated completely the computed certainty will be less than or equal to the true certainty.
2. The use of non-numeric certainties allows us to talk about consistent and inconsistent assumptions at a semantic level. Numeric systems based on probabilities, and their updating by conditionalization cannot do this.
3. Fuller use can be made of the information available to the user, namely the argument (proof tree for PROLOG) created when running a query. This allows the expert system builder to pay attention to aspects of knowledge representation, in its relation to plausible reasoning, that are not captured by approaches which assume the argument is decomposable in the sense that any subargument can be characterized by a single number. This effect is particularly noticeable when analysing or debugging results. The extra semantic information available from the non-numeric certainties allows the dependencies between various results, and the reason for them, to be clearly seen.

The interpreter of Section 3.2 (with a little elaboration) has been implemented and runs inside the YAPES expert system shell [18]. One major difference between the YAPES implementation and Fig. 3 is the incorporation of a simple mechanism for intelligent backtracking. This has the advantages of (a) minimizing the number of questions that need to be asked of the user and (b) easing the task of explanation.

(a) If we have a goal that is assumed to be true with some instantiation of variables, and a subsequent goal fails, we do not want to backtrack and ask the user whether or not the goal has another solution unless the variable binding that was assumed is or may be responsible for the failure. In this case it is worth doing some extra computation to avoid burdening the user with useless (and probably confusing) questions.

(b) A major concern with expert systems is to ensure that they can generate explanations of their behaviour. We have found that when an unnecessary question is asked (unnecessary because a subsequent goal must fail) the user can lose confidence in the system and/or become confused.

The most intriguing application of a general-purpose plausible reasoning mechanism of this kind is to the processing of inductively derived rules. It is a truism to say that the major problem with the large-scale application of knowledge-based systems technology is the process of knowledge acquisition itself. We have been interested in the inductive acquisition of such knowledge for some years. The applications we are considering at present are ones where there is a large body of existing data in machine-readable form and a need for expert analysis of such data. One such application is described in [16]; others can be seen in [25, 14, 15]. Since an inductive learning mechanism can, by definition, be wrong we need to attach some measure of plausibility to the rules that are generated. These plausibility measures generally carry much more information than can be conveyed by a mere numerical measure of probability. We need to know for instance the circumstances in which rules are applicable, and the degree to which they are applicable.

Acknowledgement

This work was supported by the Office of Naval Research under contract N00014-85-G-0243.

REFERENCES

1. Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule based expert systems.* Addison Wesley.
2. Cheeseman, P. C. (1985). In defence of probability. In *Proceedings of IJCAI-85*, pp. 1002–9, Kaufmann, Los Angeles, CA.
3. Cohen, P. R. (1985). *Heuristic reasoning about uncertainty: An artificial intelligence approach.* Pitman, London.
4. Gaifman, H. (1964). Concerning measures on first-order calculi. *Israeli Journal of Mathematics*, **2**, pp. 1–18.
5. Gaschnig, J. (1982). Application of the PROSPECTOR system to geological exploration problems. In *Machine Intelligence 10*, (eds J. Hayes, D. Michie, and L. Mikulich). Ellis Horwood, Chichester.
6. Gaschnig, J. (1980). *Development of uranium exploration models for the PROSPECTOR expert system.* Final Report, SRI International, Menlo Park, CA.
7. Good, I. J. (1963). Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables. *Annals of Mathematical Statistics*, **34**, pp. 911–34.
8. Good, I. J. (1950). *Probability and the weighing of evidence.* Hafners, New York.
9. Gordon, J. and Shortliffe, E. H. (1985). A method for managing evidential reasoning in a hierarchical hypothesis space. *Artificial Intelligence*, **26**, No. 3, 323–58.
10. Jaynes, E. T. (1957). Information theory and statistical mechanics i. *Phys. Review*, **106**, pp. 620–30.
11. de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, **28**, No. 2, 127–62.
12. Lloyd, J. W. (1984). *Foundations of logic programming.* Springer-Verlag, Berlin.
13. McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the

standpoint of artificial intelligence. In *Machine Intelligence 4* (eds B. Meltzer and D. Michie), pp. 463–502, Edinburgh University Press, Edinburgh.
14. Michalski, R. and Chilausky, R. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean diagnosis. *Policy Analysis and Information Systems*, **4**, No. 2, 125–60.
15. Michalski, R., Mozetic, I., Hong, J., and Lavrac, N. (1986). The AQ15 inductive learning system: an overview and experiments. In *Proceedings of IMAL 1986*, Université de Paris-Sud, Orsay.
16. Niblett, T. (1984). *The CLR system—an introduction.* TIRM 84-015, The Turing Institute, Glasgow.
17. Niblett, T. (1985). Judgemental reasoning for expert systems. In *Proc. IJCAI 1985*, pp. 315–17.
18. Niblett, T. (1985). *YAPES: Yet Another Prolog Expert System.* TIRM 84-008, The Turing Institute, Glasgow.
19. Pearl, J. (1982). Reverend Bayes on inference engines: a distributed hierarchical approach. In *Proc AAAI*, pp. 133–6.
20. Polya, G. (1954). *Patterns of plausible inference.* Princeton University Press.
21. Raphael, B. (1976). *The thinking computer: mind inside matter.* W. H. Freeman, San Francisco.
22. Schafer, G. (1976). *A mathematical theory of evidence.* Princeton University Press.
23. Scott, D. and Krauss, P. (1966). Assigning probabilities to logical formulas. In *Aspects of Inductive Logic* (eds P. Suppes and J. Hintikka), pp. 219–64, North-Holland, Amsterdam.
24. Sergot, M. (1983). A query-the-user facility for logic programs. In *Proc. European Conference on Integrated Interactive Computing Systems* (eds P. Dagano and E. Sandewall), North-Holland, New York.
25. Shapiro, A. and Niblett, T. (1981). Automatic induction of classification rules for a chess endgame. In *Advances in Computer Chess* (ed. M. R. B. Clarke), pp. 73–91, Pergamon, Oxford.
26. Shapiro, E. Y. (1983). Logic programs with uncertainties. In *Proc. IJCAI-83*, pp. 520–32.
27. Spiegelhalter, D. J. (1986). Synthesis of AI and Bayesian methods in medical expert systems. In *AI Methods in Statistics Seminar*, pp. 125–7, Unicorn Seminars Ltd, London.
28. Suppes, P. (1966). Probabilistic inference and the concept of total evidence. In *Aspects of Inductive Logic* (eds P. Suppes and J. Hintikka), pp. 49–65, North-Holland, Amsterdam.

6

A Note On First-Order Theories of Individual Concepts and Propositions

B. Arbab
IBM Corporation,
Los Angeles Scientific Center, USA

Abstract

Certain classes of sentences when formalized in first-order logic, may give rise to paradoxical conclusions. Within this class are sentences with one or more occurrences of such words as *know, believe, aware, discover,* and so on.

McCarthy (1979) has introduced the first order theory of individual concepts and propositions as a formal language for representing such sentences. McCarthy's formal language is intended to avoid paradoxical conclusions. This paper presents an example whose formalization in the latter language gives rise to similar paradoxical conclusions.

An alternative scheme for representation and manipulation of such sentences in first order logic, with no modal operators, is then outlined.

1. INTRODUCTION

Sentences involving knowledge, belief, and so on when formalized in first-order logic may give rise to highly counter-intuitive conclusions. For example, the two sentences:

> King George wished to know whether Sir Walter Scott is the author of Waverley. (1)

and

> Sir Walter Scott is the author of Waverley. (2)

if naïvely formalized† in first-order logic, which includes axioms of equality, will give rise to the conclusion that:

†For example, formalizing 1 as $K(G,Eq(S,A(W)))$ where K is a binary predicate symbol for *wished to know whether*, G a primitive constant for *King George*, Eq a function symbol for equals, S a primitive constant for Sir Walter Scott, A a function symbol for *Author of*, and W a primitive constant for *Waverley*. And formalizing 2 as $Eq(S,A(W))$ permits both $K(G,Eq(S,S))$ and $K(G,Eq(A(W),A(W)))$ as conclusions on the basis of the axioms of equality (that two equals can everywhere be substituted one for another).

King George wished to know whether Sir Walter Scott is Sir Walter Scott. (3)

The latter conclusion, though not contradictory, is highly counter-intuitive. As Russell (1905) put it: 'an interest in the law of identity can hardly be attributed to the first gentleman of Europe'. Following Church (1988) the above is referred to as *the paradox*† *of the name relation.*

McCarthy considers a similar example:

Pat knows Mike's telephone number. (4)

Mary has the same telephone number as Mike. (5)

to show that a normal formalization in first-order logic will give rise to the paradoxical conclusion:

Pat knows Mary's telephone number. (6)

and that such conclusions are not possible in the proposed first-order theories of individual concepts and proposition.

The paradox of the name relation was first discussed by Frege (1892)‡, whose informal solution introduced two new ideas. First, the notion of an indirect (sometimes called oblique or opaque) context which occurs whenever words like *know, believe, seek, search, necessary, possible,* and so on are used in a sentence. Second, that singular names§ have not only a denotation but also a sense‖.

At the heart of Frege's solution lies the idea that names which occur in an indirect context, shift their denotation to what was their sense in an ordinary context. Likewise, a name which occurs in a doubly indirect

†Carnap (1956) used the word *antinomy,* but the word *paradox* is preferable since no apparent contradiction occurs in the absence of any further assumptions.

‡The idea of a formal language was first conceived by Frege (1879), who was not completely satisfied with his choice of the identity relation (what today are called the axioms of equality). In his 1892 paper, he clearly shows the shortcomings of the earlier identity relation and advocates a new relation which takes into account the sense of a name as well as its denotation.

§A singular name is one which has as part of its meaning that there is only one thing denoted. Pat, for example, is a singular name that denotes a particular person. Human, however, is a general name and not singular. In a formalized language, variables play the role of general names.

‖That which two names (sentences) of languages L1 and L2 (not necessarily different) must have in common to be correct translations of one another. From an intuitive (intensional) point of view it is clear what is meant by *sense* of a name. It is not, however, at first sight clear from a computational (extensional) point of view what Frege had in mind in referring to the *sense* of a name. In fact senses are said to be more or less intensional entities and not extensional, even though no exact and general definition exists for drawing a sharp line between intensionality and extensionality. The present author has found a convenient explication through use of a computing metaphor (Arbab 1988). In this metaphor the sense of a name corresponds to a pointer (the memory location where the name is stored).

context will denote its singly indirect sense. In short, an infinite array of senses is called for. Otherwise, the paradox can be reconstructed at level $N+1$, where there are N levels of senses of a name (an example is given later in this paper).

In this paper an example is presented whose formalization in the first order theories of individual concepts and propositions will give rise to similar paradoxical conclusions. The particular example presented involves a doubly indirect context (someone knows whether someone knows something). It may very well be that McCarthy (1979) knew about the problem presented here, for his concluding remarks are as follows:

> ... Creary in particular has shown the inadequacy of the formalism for expressing all readings of the ambiguous sentence '*Pat knows that Mike knows what Joan last asserted*'. There has not been time to modify the formalism to fix this inadequacy, but it seems likely that concepts of concepts are required for an adequate treatment.

The particular problem demonstrated by Lewis Creary is unknown to the present author. The solution to the problem presented in this paper, however, does require the formalized language to permit concepts of concepts. In fact, as pointed out by Frege (1892), an infinite array of concepts is needed. Otherwise, the paradox can be reconstructed at level $N+1$, where there are N levels of concepts.

Construction of a formal language based on sense and denotation has proved to be a difficult task. Church presents three different alternatives under which a formulation of the logic of sense and denotation can be carried out. The three alternatives correspond to different sets of assumptions under which two sentences can be considered to have the same sense or express the same proposition: That two sentences S and $S1$ have the same sense (express the same proposition) if and only if $S = S1$ is *logically valid* is called *Alt(2)*. A stronger criterion or identity between senses, *Alt(1)*, is that S is convertible to $S1$ according to the rules of lambda calculus. The strongest criterion of identity between senses, *Alt(0)*, is that S and $S1$ differ at most by one or more alphabetic changes of bound variables, or one or more interchanges of synonymous† notations.

A sound system of axioms characterizing two of these alternatives, *Alt(2)* and *Alt(1)*, has been specified by Church (1946, 1973, 1987). Formulation of the logic of sense and denotation remains open under *Alt(0)*, the strongest alternative under which two sentences can be considered to express the same proposition. The solution outlined at the end of this paper is under *Alt(0)*. However, it differs from the logic of sense and denotation.

†Two constants are said to be synonymous if they have the same sense as well as denotation. They are concurrent if they have the same denotation.

2. McCARTHY'S SOLUTION

In order to lay the groundwork for reintroducing the paradox of the name relation in the first-order theory of individual concepts and propositions, some definitions and one example from McCarthy (1979) are reproduced in this section. A new example is then presented, whose analysis in the light of the definitions given leads, however, to a paradoxical conclusion.

2.1. Definitions

From the third paragraph on page 130 and conventions 1–9 of page 131 of McCarthy (1979), it is concluded that the formalized language for expressing items of knowledge has the following two linguistic domains.

1. **Real world objects** constitute a linguistic domain whose members are (or at least include) italic terms with all letters lower case. Members of this linguistic domain denote individuals in the real world. For example '*mike*' is a member of this (linguistic) domain and denotes some individual in the real world, whom (or which) we may therefore call simply *mike*. Moreover, the name '*mike*' has in addition to its quotation name also the name '*Mike*'. Thus *mike* is the denotation of the name '*Mike*', and the intention is to use the name '*Mike*' in place of the more usual quotation name '*mike*'.

2. **Concept objects** constitute a linguistic domain whose members are (or include) italic terms starting with a capital letter. Such terms are understood as denoting concepts of individuals, but the question as to what these concepts really are is left open as not being central to the theory. McCarthy does, however, remark that they may be regarded as abstract expressions, but does not explain further. For example, *Mike* is the concept associated with the name 'Mike'.

Some philosophical issues concerning the nature of the members of these two domains are left open. It is clear, however, from paragraph 3 of page 130 of McCarthy, that members of these two domains are not to be ordinary names, as in Frege (1892), which have both a denotation and a sense. Members of these two domains are capable only of denoting; not of connoting. Consider the example on page 129 of McCarthy, which was repeated in the introduction section. McCarthy avoids the paradoxical conclusion by formalizing sentence 4, as:

$$know(pat, Telephone(Mike)) \tag{7}$$

where formula 7 follows from formulas (1) and (10) of McCarthy, with function types

$$know: Realworldobjects \times Concepts \rightarrow t$$
$$Telephone: Concepts \rightarrow Concepts$$

Mike ∈ *Concepts*
pat ∈ *Realworldobjects*

The above notation is used for specifying the type of each function symbol and each predicate symbol. For example, the first argument of *know* is from the domain *Realworldobjects.* The second argument is from the *Concepts* domain and the result is the truth-value *t.* The argument of the function *Telephone* is from the *Concepts* domain, as is its result. Sentence 5 is formalized as:

$$true(Equal(Telephone(Mary), Telephone(Mike))) \quad (8)$$

This is formula (26) of McCarthy (1979) with function types

true: Concepts → *t*
Equal: Concepts × *Concepts* → *Concepts*
Telephone: Concepts → *Concepts*
Mike ∈ *Concepts*
Mary ∈ *Concepts*

It is claimed that from 7 and 8 it does not follow that

$$know(pat, Telephone(Mary)) \quad (9)$$

where 9 follows from equations (27) and (10) of McCarthy and is the formalization of the paradoxical conclusion 6. Let us examine the basis of this claim. Formula 8 seems to assert that the two concepts of Mary's telephone number and Mike's telephone number are equal to each other. If this were the case, by the axioms of equality we would be able to arrive at the paradoxical conclusion 9, by replacing *Telephone*(*Mary*) by *Telephone*(*Mike*) in formula 7. However, sentence 5 is not an assertion about the concepts of the telephone numbers of Mary and Mike, because it is not a sentence occurring in an indirect context. Sentence 5 rather, is an assertion about the two telephone numbers of Mike and Mary. Thus it is better to formalize 5 as

$$equal(denot(Telephone(Mary)), denot(Telephone(Mike))) \quad (10)$$

where function types are as follows:

equal: Realworldobjects × *Realworldobjects* → *t*
denot: Concepts → *Realworldobjects*
Telephone: Concepts → *Concepts*
Mike ∈ *Concepts*
Mary ∈ *Concepts*

The well-formed formula 9 does not follow from 7 and 10 because formula 10 expresses an equality between the denotations of *Telephone*(*Mike*) and *Telephone*(*Mary*). On the other hand, 7 is a relation

between *pat* and *Telephone*(*Mike*). From what two formulas can formula 9 be obtained as a conclusion? Consider the sentence

Pat knows whether Mike's telephone number is the same as Mary's telephone number (11)

which is formalized as

true(*K*(*Pat*,*Equal*(*Telephone*(*Mary*),*Telephone*(*Mike*)))) (12)

This is formula (28) of McCarthy. According to him, 7 and 12 plus suitable axioms about knowledge will allow 9 as a conclusion.

2.2. New Example

Consider now the following example in which formalization in the theory proceeds analogously to that of the above example. The paradoxical conclusion, however, cannot be avoided. The two sentences are:

Pat knows whether Pythagoras knows whether the morning star is the evening star (13)

and

Pat knows whether the morning star is the evening star (14)

the paradoxical conclusion which cannot be avoided is:

Pat knows whether Pythagoras knows whether the morning star is the morning star (15)

Sentence 13 should be formalized in such a way that *Pat*, *Pythagoras*, *Mstar*, and *Estar* are concepts associated with Pat, Pythagoras, the morning star and the evening star.

true(*Know*(*Pat*,*Know*(*Pythagoras*,*Equal*(*Mstar*,*Estar*)))) (16)

where types are as follows:

Know: Concepts × *Concepts* → *Concepts*

Equal: Concepts → *Concepts*

Pat,*Pythagoras*,*Mstar*,*Estar* ∈ *Concepts*

Sentence 14 would have to be formalized as

know(*pat*,*Equal*(*Mstar*,*Estar*)) (17)

From 16 and 17 and the suitable axioms of knowledge as used in the example on page 129 of McCarthy we can conclude that

true(*Know*(*Pat*,*Know*(*Pythagoras*,*Equal*(*Mstar*,*Mstar*)))) (18)

This is the well-formed formula corresponding to the paradoxical conclusion 15. This conclusion can be reached on the ground that 16 expresses a relation between *Mstar* and the *Estar* and 17 expresses that Pat knows whether the two are equal. Thus, he claims, by the axioms of equality and the suitable axioms of knowledge, the two can be interchanged in any well-formed formula.

Sentence 15 is not a conclusion of 13 and 14 according to Frege's (1892) original solution. Unlike McCarthy's first-order theories of first-order propositions and concepts, Frege allows an infinite hierarchy of concepts of names. Furthermore, every time a name occurs at level N of an indirect context, it will automatically shift its denotation to what was the sense of the same name at a level $N-1$ indirect context. Level zero of indirect context is of course the same as the ordinary context for names. In 13 *the morning star* and *the evening star* occur in a doubly indirect context, once in the context of Pythagoras knowing, and again in the context of Pat knowing.

The substitution of *the morning star* for *the evening star* in sentence 13 is not allowed on the ground that the two names denote two different objects. In 14 the names *the morning star* and *the evening star* occur in a singly direct context and thus denote the sense of the same names in an ordinary context. Whereas in 13 the two names occur in a doubly indirect context and thus denote the sense of the same names in a singly indirect context. Note that the sense of a name in a singly indirect context differs from the sense of that name in an ordinary context.

3. PROPOSED SOLUTION

Ajdukiewicz's (1960) solution to the paradox of the name relation revolves around the ambiguity of sentences similar to 4. It is the process of removing the ambiguity of these sentences that leads to the solution. In his attempt to find an extensional solution to the paradox of the name relation, Ajdukiewicz introduces the novel idea of abstraction with respect to a constant. The notion of abstraction with respect to a variable is familiar in most formal languages, but not abstraction with respect to a constant.

Church (1988) has formalized the solution informally explained by Ajdukiewicz. The idea of a proposition surrogate is one that has arisen out of this formalization. In this section, Church's formal notation along with Ajdukiewicz's informal explanation is applied to some examples of the paradox of the name relation. It is shown how the paradoxical conclusions are avoided.

Consider McCarthy's example that was discussed previously. According to Ajdukiewicz, sentence 4 is an ambiguous sentence. It has two different meanings. One meaning leads to the paradox, but the other does

not. Thus, it should be no surprise that most people would understand the meaning of sentence 4 that does not lead to the paradox, and not the other that leads to the paradox. The first meaning of sentence 4 is:

Pat knows
about a particular number,
about the equality relation,
about Mike,
about the function telephone number,
that the telephone number of Mike is equal to
the particular number.

The second meaning is:

Pat knows
about a particular number,
about the equality relation,
about the telephone number of Mike,
that the telephone number of Mike is equal to
the particular number.

In the second meaning, Pat does not explicitly know about the function telephone number or Mike, but only that the result of application of the function telephone number to the argument Mike is some particular number. It is this meaning of 4 that leads to the paradoxical conclusion and not the first meaning.

In what follows, both meanings of 4 are formalized according to the notation introduced by Church (1988). It is then shown how the second meaning leads to the paradox, whereas the first meaning avoids it.

The first meaning of 4 is formalized under Alt(1) as:

$$know(pat, \langle \lambda D \lambda F \lambda G \lambda B.D(\lambda X.F(x, G(B))), \iota, equal, telephone, mike \rangle) \quad (19)$$

and under Alt(0) as:

$$know(pat, \langle \lambda D \lambda F \lambda G \lambda B \langle D, \lambda X \langle F, X, \langle G, B \rangle \rangle \rangle, \iota, equal, telephone, mike \rangle) \quad (20)$$

where ι is denoting the description operator, not contextually defined, and *telephone* and *equal* are function symbols, *mike* and *pat* are primitive constants in the language corresponding to Mike and Pat; D, F, G, B, and X are bound variables. The first argument of *know* is a primitive constant and the second argument is an ordered n-tuple. This ordered n-tuple is referred to as a proposition surrogate and is the object of belief of the first argument. The first member of a proposition surrogate must be of the form $\lambda X1 \lambda X2 \ldots \lambda Xn.M$ where M has as its only free variables exactly one free occurrence of each of the variables $X1, X2, \ldots, Xn$. The remaining members of the proposition surrogate must be constants

$C1, C2, \ldots, Cn$ that are of the same type as the variables $X1, X2, \ldots, Xn$ but not necessarily all different.

The first member of the proposition surrogate gives only the form. The constants, names, and functions are then abstracted and listed separately. The original formula can always be obtained from the proposition surrogate by applying the first member to the rest of the members. This device is used primarily to distinguish between application of a function to its arguments and the resulting value. For example, $5 = \sqrt{25}$ is true, whereas $5 = \langle \lambda F \lambda X \rangle F, X \rangle, \sqrt{}, 25\rangle$ is not true. However, applying the first member of $\langle \lambda F \lambda X \rangle F, X \rangle, \sqrt{}, 25\rangle$ to the other members will result in $\langle \sqrt{}, 25\rangle$; another application yields $\sqrt{25}$ which is equal to 5.

The second meaning of 4 is formalized under Alt(1) as:

$$know(pat, \langle \lambda D \lambda F \lambda A D (\lambda X.F(X,A)), \iota, equal, telephone(mike)\rangle) \quad (21)$$

and under Alt(0) as:

$$know(pat, \langle \lambda D \lambda F \lambda A \langle D, \lambda X \langle F, X, A\rangle\rangle, \iota, equal, telephone(mike)\rangle) \quad (22)$$

Note that in 22 Pat does not know about the function *telephone* or *mike*, whereas 20 has constants corresponding to both *mike* and the function *telephone* listed explicitly within the constant list occurring in the proposition surrogate.

Now, sentence 5 is formalized in the usual way as:

$$telephone(mike) = telephone(mary). \quad (23)$$

It is possible to replace $telephone(mike)$ by $telephone(mary)$ in 22, on the grounds of 23 and the axioms of equality, thus arriving at

$$know(pat, \langle \lambda D \lambda F \lambda A \langle D, \lambda X \langle F, X, A\rangle\rangle, \iota, equal, telephone(mary)\rangle) \quad (24)$$

which is the well-formed formula corresponding to the paradoxical conclusion 6. No such replacement is possible, however, in 20 on the grounds that $telephone(mike)$ is not directly identifiable in the list of constants of the proposition surrogate. Indeed, 20 is the intended and the understood meaning of sentence 4.

The solution as outlined above does not address the problem with respect to primitive constants of the language. For example, the two sentences *Pythagoras knows whether the morning star is the evening star* and *the morning star is the evening star* when formalized according to the above theory of proposition surrogates will give rise to the valid (but boring) conclusion *Pythagoras knows whether the morning star is the morning star.* For the full treatment and addition of proposition surrogates to programming languages the reader is referred to Arbab (1988).

4. SUMMARY

Various solutions to the paradox of the name relation are examined. McCarthy's (1979) solution to the paradox is shown defective by means of a new example. Frege's (1892) solution to the paradox is also introduced and applied to the examples in this paper. Formalization of Frege's solution to the paradox, under Alt(0), is still an open problem. Finally, following Church (1988) and Ajdukiewicz (1960), a partial solution to the paradox is introduced and applied. For the complete solution see Arbab (1988). This solution is particularly suited for automatic representation of sentences that contain words which introduce an indirect context.

Acknowledgements

The idea of a proposition surrogate was first developed by Church (1988). This development of proposition surrogates started from a short abstract by Ajdukiewicz (1960), in which a solution to the paradox of the name relation was only informally outlined. Ajdukiewicz, however, had outlined in a later paper (1967) an idea that upon inspection is very close in content to that of proposition surrogates. The problem with respect to primitive constants was not addressed by Ajdukiewicz until 1967. In this paper he clearly shows that analysis of sentences containing primitive constants according to his technique, which is very similar to proposition surrogates, will admit substitutions of primitive constants for one another. He then argues that if the resulting conclusions seem to be counter-intuitive it must be that we understand such sentences as metasentences and not as object language sentences. The present author fails to understand this particular philosophical view taken by Ajdukiewicz. The above example is simply another instance of the paradox of the name relation which a proposed solution should resolve.

The author is grateful to Professors Alonzo Church, Donald Michie and Stott Parker for their review and comments on this paper. Needless to say, all remaining errors are attributable solely to the author.

REFERENCES

Ajdukiewicz, K. (1967). Intensional expressions. In *Kazimierz Ajdukiewicz. The scientific world-perspective and other essays 1931–63*, (ed. J. Giedymin) pp. 320–47, D. Reidel Publishing Company.

Ajdukiewicz, K. (1960). A method of eliminating intensional sentences and sentential formulae. In *Atti del XII Congresso Internazional di Filosofia*, pp. 17–24.

Arbab, B. (1988). *A formal language for representation of and reasoning about indirect context.* PhD thesis, University of California at Los Angeles

Carnap, R. (1956). *Meaning and necessity.* The University of Chicago Press, 2nd edition.

Church, A. (1946). A formulation of the logic of sense and denotation. *The Journal of Symbolic Logic*, **11** No. 1. Abstracts of Papers.

Church, A. (1973). Outline of a revised formulation of the logic of sense and denotation (Part I) *Nous*, **VII(I)**, 24–33.

Church, A. (1987). Intensionality and the paradox of the name relation. In *Themes from Kaplan*, Stanford Press. The content of this paper was presented as an invited lecture at a joint symposium of the A.P.S. and the Association for Symbolic Logic in Berkeley, California, March 1983.

Frege, G. (1879). Begriffsschrift, a formal language, modelled upon that of arithmetic, for pure thought. In *From Frege to Godel*, (ed. J. van Heijenoort) pp. 1–82, Harvard University Press, 1977.

Frege, G. (1892). On sense and meaning. In *The Philosophy of Language*, (ed. A. P. Martinich) pp. 212–20, Oxford University Press, 1985. Originally published in 1892.

McCarthy, J. (1979). First order theories of individual concepts and propositions. In *Machine Intelligence 9*, (eds J. Hayes, D. Michie, and L. I. Mikulich) pp. 120–47, Ellis Horwood.

Russell, B. (1905). On denoting. *Mind*, **14**, pp. 479–93.

INDUCTIVE FORMATION OF PROGRAMS AND DESCRIPTIONS

7

Inverting the Resolution Principle

S. H. Muggleton
The Turing Institute,
Glasgow, UK

Abstract

In this paper we describe the current status of an ongoing research project investigating a novel form of Machine Learning in which the learner's vocabulary is enriched by the machine suggesting useful new descriptive terms for the user to accept or reject. An algorithm called Duce has been shown to be effective along these lines in developing and extending propositional theories within a chess endgame domain and a diagnostic domain of neuro-psychology. By showing that Duce's transformational operators are based on reversing the steps of a resolution proof we show that Duce's learning method is sufficient for learning any propositional theory.

1. INTRODUCTION

Duce [4] is an algorithm which produces hierarchical concept descriptions from large numbers of examples. Whereas the ID [7] and AQ [3] families of inductive algorithms require all necessary attributes to be provided before learning can take place, Duce develops new attributes by incrementally building them from existing ones, testing each against the user for comprehensibility. Duce uses a set of transformations of propositional Horn clauses which successively compress the example material on the basis of generalizations and the addition of new terms. In the following descriptions of three of the six Duce operators, lower-case Greek letters stand for conjunctions of propositional symbols.

1. *Intra-construction* This is the distributive law of Boolean equations. We take a set of rules such as

   $h_1 \leftarrow \alpha\beta$
   $h_1 \leftarrow \alpha\gamma$

   and replace them with the rules

   $h_1 \leftarrow \alpha h_3$
   $h_3 \leftarrow \beta$
   $h_3 \leftarrow \gamma$

The user either names the new concept h_3 or rejects it.

2. *Absorption* This operator is from Sammut and Banerji [10]. Given a set of rules, the body of one of which is completely contained within the bodies of the others, such as

   $h_1 \leftarrow \alpha\beta$
   $h_2 \leftarrow \alpha$

   one can hypothesize

   $h_1 \leftarrow h_2\beta$
   $h_2 \leftarrow \alpha$

   The user can either accept this generalization or reject it.

3. *Identification* This operator has preconditions which are stronger than those of intra-construction. A set of rules which all have the same head, the body of at least one of which contains exactly one symbol not found within the other rules, such as

   $h_1 \leftarrow \alpha\beta$
   $h_1 \leftarrow \alpha h_2$

   can be replaced by

   $h_1 \leftarrow \alpha h_2$
   $h_2 \leftarrow \beta$

   Again the user can either accept this generalization or reject it.

Duce uses the compaction of the rulebase produced by each of the six operators to guide the search for the next operator to apply. In [4] we give the set of characteristic formulas, one for each operator, which predict the exact *symbol reduction* produced by each operator, the number of *symbols* in a rule being equal to the rule-body length plus one for the rule-head. Since Duce applies only operators which give a positive symbol reduction it can easily be shown that the algorithm terminates after a finite number of operator applications.

2. APPLICATION DOMAINS

2.1. KPa7KR application

The first large-scale test of Duce's capabilities [4], was an attempt to reconstruct automatically Shapiro and Kopec's expert system [11] for deciding whether positions within the endgame of King-and-Pawn-on-a7 v. King-and-Rook were won for white or not. A set of 3196 examples was used, and Duce's questions were answered by the chess experts Ivan

Bratko and Tim Niblett. The result was a comprehensible restructuring of the domain, topologically similar to Shapiro and Kopec's original structure, though an order of magnitude more bulky.

2.2 Neuro-psychology application

A second, and previously undescribed structuring experiment was carried out by the author using Duce at Interact Corporation, Canada. In this Duce was used to construct a problem decomposition for deciding on dysfunction of the left parietal brain area of children with learning disabilities. The input to the algorithm consisted of 227 diagnosed cases. Each case contained the results of a battery of approximately 100 binary-valued clinical tests. Each case was marked with a diagnosis of normal/abnormal left parietal lobe by the resident clinical neuro-psychologist, Dr Russell. Using these cases Duce carried out an interactive session in which Russell was asked to answer a total of 53 questions. During and subsequent to the construction of the rulebase, a set of 48 independent cases was used to test the performance of the new rule-set. Since the cases and generated rules were inherently noisy, a majority-vote mechanism was used for rule evaluation. After all 53 questions had been answered, 43 of the 48 test cases agreed with Russell's diagnosis, in other words, 90 per cent agreement. In contrast, an existing expert system developed by Russell had only a 63 per cent agreement rate with Russell's diagnoses over the same test data. While Duce's structured rulebase took 2–3 person-days to build and verify, the equivalent part of the hand-built expert system is conservatively estimated to have taken 2–3 person-months to generate, improve, and verify.

In parallel with the supervised construction of the Duce rulebase, Duce was run on the same cases in unsupervised mode. In this mode, all generalization questions were answered affirmatively and all new concepts were arbitrarily named. Performance in unsupervised mode stabilized after 27 questions to a level of 25 per cent agreement with Russell's diagnoses of the same test cases.

Unlike the endgame experiment in which an exhaustive example set was used, the neuro-psychological example set was relatively sparse. As a consequence, whereas no rejections were necessary in the case of the chess experiment, an average of 10 rejections were required per acceptance with the neuro-psychological data. This seems to indicate the need for expert supervision of Duce where sparse data is involved, and explains the dramatic difference in verification results between the supervised and unsupervised data.

The structure of the rulebase created by Russell working with Duce is shown below. This hierarchical structure contains groups of rules associated with each node of the network. The sub-types implied by this

hierarchy were, according to Russell, 'clinically significant', and relate directly to neuro-psychological sub-types based on Wide-range-achievement-test (WRAT) results in arithmetic, reading, and spelling.

```
LPAb_GoodV
LPAb_PoorA1
  LPAb_PoorA
LPAb_BadA1
  LPAb_BadA1_RS_AST
  LPAb_BadA
    LPAb_BadA_PoorS
      LPAb_BdA_PrS_BadAST_GdSSPT
    LPAb_BadA_BadAST
      LPAb BadA_AST_S1
        LPAb_BadA_AST_S
        LPAb_BadA_AST_S1_V
```

3. THEORY

Duce has shown a considerable amount of sucess within the application domains described in the previous section. However, there are a number of questions concerning the methodology employed within Duce which are of interest both from a theoretical and a practical viewpoint.

1. *Completeness of operators* Six operators are used by Duce to carry out generalizations and introduce new terms. How complete are these? Are they sufficient to learn any arbitrary set of propositional clauses given enough examples? (See Section 3.2.)
2. *Search* Duce presently searches through the set of conjunctions of predicate symbols to find which operator to apply next. For this reason Duce can be myopic in its choice of new terms. The discrepancy in complexity between Duce's solution and human solutions (see Section 2.1) seems to indicate that this problem can be quite severe. New methods of searching for operators are required. (See Section 3.3.)
3. *Extensions to first-order representation* Bain [2] has described a failed attempt to use Duce to learn a simple chess definition of position legality from only positional attributes. Although the definition can be described simply in first-order predicate calculus the cumbersomeness of Bain's description is not eased by adding extra propositional attributes. This gives incentive to an investigation into extending the Duce approach to deal with first-order predicate calculus (see [5]).

3.1 Inverting resolution

Although it is apparent to many researchers in Machine Learning that there is a strong relationship between deductive theorem-proving mechanisms and inductive inference, this idea has rarely been investigated to any greater depth than to notice that the ideas of *logical subsumption* or *logical implication* are central to both. One exception to this is Plotkin [6] who investigated the idea that 'just as unification was fundamental to deduction, so might a converse be of use in induction.'

From this idea Plotkin went on to develop the concept of *least general generalization*, or *anti-unification* of literals and clauses.

Unification is a basic idea within Robinson's [9] theory of resolution. Another important concept within this theory is that of the resolution tautology, or rule of inference. As Plotkin [6] notes, 'It is interesting that ... the similarity between induction and deduction breaks down ... [with anti-unification]. What is useful is not a concept of unification of two clauses, but the deduction principle called resolution.'

We now show that the analogy between deduction and induction can be extended fruitfully, and that in fact the operators used by Duce are merely the inverse of resolution. In a later section, this fact will lead us to a proof of the sufficiency of the Duce operators.

In this section we limit our discussion of resolution to binary resolution of propositional Horn clauses. However, in [5] we extend this analysis to deal with first-order representations. Let C_1 and C_2 be the two clauses

$$C_1 = (h_1 \leftarrow \alpha h_2)$$

$$C_2 = (h_2 \leftarrow \beta)$$

We write the resolvent or resolved product of C_1 and C_2 as

$$C = C_1 \cdot C_2 = (h_1 \leftarrow \alpha\beta)$$

We now define the resolved quotient as follows

$$C_1 = C/C_2$$

Alternatively, the author calls C_1 the *identificant* of C and C_2. Similarly

$$C_2 = C/C_1$$

Again we call C_2 the *absorbant* of C and C_1.

It is now straightforward to define the absorption and identification operators described informally in Section 1.

Definition 1 Given a propositional Horn clause program $P \supseteq \{C, C_1\}$, the absorption operator, *Abs*, transforms P to $P' = (P - \{C\}) \cup \{C/C_1\}$.

Definition 2 Given a propositional Horn clause program $P \supseteq \{C, C_2\}$, the identification operator, *Ident*, transforms P to $P' = (P - \{C\}) \cup \{C/C_2\}$.

We can also define the inverse of both of these operators uniquely.

Definition 3 Given a propositional Horn clause program $P \supseteq \{C_1, C_2\}$, the inverse absorption operator, Abs^{-1}, transforms P to $P' = (P - \{C_2\}) \cup \{C_1 \cdot C_2\}$.

Definition 4 Given a propositional Horn clause program $P \supseteq \{C_1, C_2\}$, the inverse identification operator, $Ident^{-1}$, transforms P to $P' = (P - \{C_1\}) \cup \{C_1 \cdot C_2\}$.

We now give a formal definition of the intra-construction operator of Section 1 and its inverse. Let $A = (h_3 \leftarrow \gamma h_4)$, $BB = \{B_1, \ldots, B_n\} = \{(h_4 \leftarrow \delta_1), \ldots, (h_4 \leftarrow \delta_n)\}$, $CC = \{C_1, \ldots, C_n\} = \{(A \cdot B_1), \ldots, (A \cdot B_n)\} = \{(h_3 \leftarrow \gamma\delta_1), \ldots, (h_3 \leftarrow \gamma\delta_n)\}$.

Definition 5 Given a propositional Horn clause program $P \supseteq CC$, the intra-construction operator, *Intra*, transforms P to $P' = (P - CC) \cup \{A\} \cup BB$.

Definition 6 Given a propositional Horn clause program $P \supseteq (\{A\} \cup BB)$, the inverse intra-construction operator, $Intra^{-1}$, transforms P to $P' = (P - (\{A\} \cup CC)) \cup BB$.

Lemma 1 If the program transformation $P \rightarrow P'$ is carried out by either Abs^{-1}, $Ident^{-1}$, or $Intra^{-1}$ then P subsumes P'.

Proof Follows from the fact that all these operators replace more general clauses with more specific ones. □

The reader may wonder how A and BB are constructed in the definition of *Intra*. As a special case of Plotkin's [6] *least general generalization* (*lgg*) of clauses, we say that γ is the *lgg* of the bodies of clauses within CC ($bodies(CC)$) if and only if γ is the common intersection of propositional symbols of $bodies(CC)$. Given γ and a new predicate symbol h_3, it is straightforward to construct A and BB. It is only through reversal of multiple resolution steps that the introduction of new predicate symbols becomes possible.

3.2. Completeness of Duce operators

In order to ensure that the success of the Duce applications described in Section 2 was not due to some peculiar property of the domains involved we need to show that the operators used by Duce are sufficient to learn any arbitrary set of propositional clauses. Clearly we need to specify some restriction on the allowable forms of examples used, otherwise Duce could merely be presented with any desired solution as its input. Let P be an arbitrary target propositional Horn clause program. The vocabulary used in P ($vocab(P)$) is then simply the set of propositional symbols in P. The primitive vocabulary of P ($prim(P)$) is the set of

symbols not defined in terms of other predicate symbols. Thus $prim(P) = vocab(P) - heads(P)$, where $heads(P)$ is the set of clause heads of clauses within P. Now let E be a set of example propositions from which P can be learned by Duce. A reasonable restriction on allowable forms of examples used would seem to be that

(1) P subsumes E, i.e. any statement which can be derived from E can also be derived from P;

(2) each clause body in $bodies(E)$ is composed only of predicate symbols from $prim(P)$;

(3) the vocabulary of P is an extension of that of E, i.e. $vocab(P) - vocab(E) \neq \{\}$.

If these conditions are met then we will say that E is a *legitimate* example set of P. First we define the abstract algorithm $Duce_{(Abs,Intra)}$ which is a non-deterministic version of the Duce algorithm limited to using only the *Abs* and *Intra* operators. In the following an inverse derivation $E \rightarrow P_1 \rightarrow \ldots \rightarrow P_n$ is a mixed sequence of absorption and intra-construction transformations of the example set E into the propositional Horn clause program $P = P_n$.

Definition 7 The algorithm $Duce_{(Abs,Intra)}(E)$ returns a set of possible Horn clause programs $H = \{P: P \text{ is an inverse derivation of } E\}$.

We can see H as being the hypothesis space of an algorithm which returns a single hypothesis. Angluin [1] introduced the notion of a *characteristic sample* set of examples for language L as being a set of examples which are sufficient to allow the inference of L. Here we use the term somewhat loosely to define a set of examples which induces a hypothesis space containing a given logic program P. If we can show that for any arbitrary propositional Horn clause program P we can generate a characteristic sample set, it follows that there is a sample set from which any P can be induced. This in turn would show that given a large enough set of examples, which in the limit must contain a characteristic sample, the Duce operators are sufficient to learn any propositional Horn clause program.

Definition 8 Given a propositional Horn clause logic program P we say that E is a characteristic sample of P for algorithm $Duce_{(Abs,Intra)}$ if and only if E is a legitimate example set of P and $P \in Duce_{(Abs,Intra)}$.

Before showing how to construct a finite characteristic sample for any logic program we will introduce the auxiliary notion of an isolated reference.

Definition 9 The clause $(h \leftarrow ap) \in P$ is said to reference predicate symbol p.

Definition 10 The clause $C \in P$ contains an isolated reference to predicate symbol p if and only if C is the only clause within P which references p.

Remark 1 If Abs^{-1} is applied to $P \supseteq \{C_1, C_2\}$ to produce $P' = P - \{C_2\} \cup \{C_1 \cdot C_2\}$ then the operator Abs^{-1} reduces the number of clauses which reference predicate symbol $p \in vocab(P)$ by one.

Remark 2 If $Intra^{-1}$ is applied to $P \supseteq (A \cup BB)$ to produce $P' = P - (\{A\} \cup BB) \cup CC$, where $A = (h \leftarrow \alpha p)$ contains an isolated reference to p, $BB = \{(p \leftarrow \beta_1), \ldots, (p \leftarrow \beta_n)\}$ is the set of all clauses containing the predicate symbol p in their heads and $CC = \{(h \leftarrow \alpha\beta_1), \ldots, (h \leftarrow \alpha\beta_n)\}$ then the program P' does not contain the predicate symbol p.

The following algorithm $Char_{(Abs,Intra)}$ can be used to generate a characteristic sample of a given propositional Horn clause logic program P.

algorithm $Char_{(Abs,Intra)}(P)$
  let $i = 0, P_0 = P$
  until P_i is a legitimate example set of P **do**
    if there is an A in P_i such that A contains an isolated reference to p
      P_{i+1} is the result of applying $Intra^{-1}$ to remove p in P_i
    else
      P_{i+1} is the result of applying Abs^{-1} to remove reference A in P_i
    let $i = i + 1$
  done
  let $f = i$
  E is P_f
  return(E)
end *Char*

Now we must show that this algorithm will generate a characteristic sample for any propositional Horn clause logic program.

Theorem 1 $Char_{(Abs,Intra)}(P)$ returns a characteristic sample for any propositional Horn clause logic program P.

Proof Let $E = Char_{(Abs,Intra)}(P)$. According to definition 8 E is a characteristic sample of P for $Duce_{(Abs,Intra)}$ if and only if E is legitimate and $P \in Duce_{(Abs,Intra)}(E)$. Let us assume that E is not a characteristic sample of P.

We will first look at the case in which the **until** loop in $Char_{(Abs,Intra)}$ terminates. According to the loop termination condition, P_f must be a legitimate example set of P. Since each step i in the derivation $P \rightarrow \ldots$

$\rightarrow P_f$ was carried out by either Abs^{-1} or $Intra^{-1}$ it follows that the sequence of transformations $(E = P_f) \rightarrow \ldots \rightarrow P$ is an inverse derivation of P from E. It follows from definition 7 that $P \in Duce_{(Abs,Intra)}(E)$, and thus E is a characteristic sample of P. We must therefore assume that the **until** loop does not terminate.

Let p be some predicate symbol in $vocab(P) - prim(P)$. By definition there must be clauses which reference p in P. These references will be reduced one by one by the **else** statement (Remark 1), with the last reference being removed by the **if** statement, together with all remaining occurrences of p in P (Remark 2). Referenced predicate symbols will be removed one by one until only unreferenced predicate symbols remain for some P_j. From repeated application of Lemma 1, P subsumes P_j. Moreover the vocabulary of P_j will have been successively reduced from that of P by applications of $Intra^{-1}$ (Remark 2). Thus by definition P_j is legitimate and the **until** loop will terminate with $f = j$. This contradicts the assumption and completes the proof. □

We now investigate the size of the characteristic sample set for a given propositional Horn clause logic program.

Theorem 2 Let $E = Char_{(Abs,Intra)}(P)$ and Ps be the set of referenced predicate symbols in P. The size of the characteristic sample set $|E| = |P| - |Ps|$.

Proof From definition 2 Abs^{-1} applies the transformation $P' = P - C_2 \cup C$, and therefore $|P'| = |P|$. From definition 4, $Intra^{-1}$ applies the transformation $P' = (BB \cup \{A\}) \cup CC$, where $|BB| = |CC|$. It follows that for $Intra^{-1}$, $|P'| = |P| - 1$. In the proof of Lemma 1 we have shown that referenced predicate symbols are removed one by one using $Intra^{-1}$. All other transformations employ Abs^{-1}. Since there must therefore be $|Ps|$ applications jof $Intra^{-1}$ it follows that $|E| = |P| - |Ps|$. □

Thus not only have we shown that the operators *Abs* and *Intra* are sufficient to learn any arbitrary propositional program but also, surprisingly, fewer examples are needed to induce a propositional program than there are clauses in that program. This is counter-intuitive to the normal belief in inductive knowledge engineering, in which we expect to use a large number of examples to induce a small number of rules.

3.3. Search: Duce macro-operators

Duce presently searches through the set of conjunctions of predicate symbols to find which operator to apply next. For this reason Duce can be myopic in its choice of new terms. The discrepancy in complexity between Duce's solution and human solutions (see Section 2.1) seems to indicate that this problem can be quite severe. However, by considering the characteristic set generating algorithm of Section 3.2 as being an

inverted strategy for propositional program construction, we have discovered a simple and effective method of improving Duce's present search mechanism. The argument is as follows. The algorithm $Char_{(Abs,Intra)}$ removes intermediate concepts (predicate symbols) one at a time. For every predicate symbol p removed, $Char_{(Abs,Intra)}$ applies Abs^{-1} repeatedly to remove all but the last reference to p. p is finally removed from the vocabulary by application of $Intra^{-1}$. In reverse this strategy becomes

(1) introduce p using *Intra*

(2) apply *Abs* to all clauses with head p.

This can be viewed as a form of macro-operator. Attempts are presently being made to implement this and other macro-operators within Duce. One severe impediment to the approach has been that whereas the symbol reduction effect of the old Duce operators can be efficiently and accurately computed using the characteristic equations of [4], no efficient method of computing the exact effect of macro-operators has been found outside application and measurement. It is estimated that application and measurement would slow the execution of the Duce algorithm by a factor of 1000 on applications the size of the KPa7KR experiment. However, various methods of approximating the evaluation have been tried, the most effective of which led to a 20 per cent compaction of the KPa7KR result [4].

4. DISCUSSION

Although much progress has been made in applying Duce to various problem domains, the present aim of our research is to extend Duce's capabilities. For these purposes it has been necessary to work out the theory underlying the Duce approach in more detail. In so doing we have discovered that Duce is a form of inverse resolution theorem prover. Many useful insights into possible improvements and extensions of the Duce algorithm have resulted from this, some of which are described in Section 3.

The two most interesting adaptions of Duce seem to lie in the directions of changing the underlying knowledge representation to first-order predicate calculus and dealing with noisy data. Although other authors have looked at related problems [8, 12, 10], all such attempts have dealt with learning single predicates, none with the more difficult problem of automatic vocabulary extension.

Acknowledgements

This paper describes work which was funded in part by the British Government's Alvey Logic Database Demonstrator. Research facilities were provided by the Turing Institute,

Glasgow, UK, Interact R&D Corporation, Victoria, BC, Canada and the US Army Office of Research in the Behavioral and Social Sciences through their Science Coordination Office in London.

REFERENCES

1. Angluin, D. (1982). Inference of reversible languages. *JACM*, **29**, pp. 741–65.
2. Bain, M. (1987). *Specification of attributes for computer induction.* TIRM, The Turing Institute, Glasgow.
3. Michalski, R. and Larson, J. (1978). *Selection of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11.* UIUCDCS-R 78-867, Computer Science Department, Univ. of Illinois at Urbana-Champaign.
4. Muggleton, S. H. (1987). Duce, an oracle based approach to constructive induction. In *IJCAI-87*, pp. 287–92, Kaufmann.
5. Muggleton, S. H. (1987). *Towards constructive induction in first-order predicate calculus.* TIRM, The Turing Institute, Glasgow.
6. Plotkin, G. D. (1971). *Automatic methods of inductive inference.* PhD thesis, Edinburgh University.
7. Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In *Introductory Readings in Expert Systems* (ed. D. Michie), pp. 33–46, Gordon and Breach, London.
8. Quinlan, J. R. (1986). Learning from noisy data. In *Machine Learning Volume 2* (eds R. Michalski, J. Carbonnel, and T. Mitchell), Kaufmann, Palo Alto, CA.
9. Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *JACM*, **12** No. 1, 23–41.
10. Sammut, C. and Banerji, R. B. (1986). Learning concepts by asking questions. In *Machine Learning: An Artificial Intelligence Approach* (eds R. Michalski, J. Carbonnel, and T. Mitchell), **2**, pp. 167–92, Kaufmann, Los Altos, CA.
11. Shapiro, A. D. (1987). *Structured induction in expert systems.* Turing Institute Press in association with Addison-Wesley.
12. Shapiro, E. Y. (1982). *Algorithmic program debugging.* PhD thesis, Yale University.

8

Non-monotonic Learning

M. Bain and S. H. Muggleton
The Turing Institute,
Glasgow, UK

Abstract

This paper addresses methods of specializing first-order theories within the context of incremental learning systems. We demonstrate the shortcomings of existing first-order incremental learning systems with regard to their specialization mechanisms. We prove that these shortcomings are fundamental to the use of classical logic. In particular, minimal 'correcting' specializations are not always obtainable within this framework. We propose instead the adoption of a specialization scheme based on an existing non-monotonic logic formalism. This approach overcomes the problems that arise with incremental learning systems which employ classical logic. As a side-effect of the formal proofs developed for this paper we define a function called 'deriv' which turns out to be an improvement on an existing explanation-based-generalization (EBG) algorithm. PROLOG code and a description of the relationship between 'deriv' and the previous EBG algorithm are described in an appendix.

1. INTRODUCTION

1.1. Motivation

Generalization is not everything in learning. New experience can often require the specialization of over-general beliefs. For example imagine that you believed that all birds fly. We might write this as

$$Flies(x) \leftarrow Bird(x)\dagger \tag{1}$$

If you were told that although an emu is a bird it cannot fly you would have to specialize (1) to deal with this exception. In this paper we discuss various ways in which this could be done. In the following we will assume that an *incremental* learning algorithm receives new examples one at a time, its belief set being revised after each example. The belief set is generalized when it does not cover a new example. On the other hand it is specialized when contradicted by a new example. Most work in machine learning is based on the related notions of generalization

†The logic notation used in this paper is similar to that of Chang and Lee [1].

and specialization. However, most treatment of the topic of specialization has been within the context of non-incremental learning algorithms such as ID3 [14], AQ11 [6], INDUCE [7], and Version Space algorithms [8]. In this paper we investigate the less explored topic of incremental specialization. Clearly an incremental learning algorithm as described above changes the coverage of its beliefs *non-monotonically*, hence the title of this paper. However, in Section 1.3 we show that the developers of logic-based machine-learning algorithms to date have avoided non-monotonic logic representations. This leads to various problems. In Section 2 we prove that it is not possible in general to preserve correct information when incrementally specializing within a classical logic framework. In Section 3 we demonstrate that this impasse can be easily avoided by learning algorithms that employ a non-monotonic knowledge representation.

Muggleton and Buntine [11] have described an algorithm called CIGOL which learns unrestricted first-order Horn clause theories on the basis of unit clause examples presented one at a time. CIGOL as described in [11] is not incremental in the sense of the description above. Although it generalizes from positive examples it does not specialize when it encounters negative examples which contradict its theory. In practice this has led to difficulties. As reported in [10], by over-generalizing CIGOL managed to outstrip the performances of both humans and propositional learning algorithms when incrementally learning a chess concept from randomly selected examples. However, having over-generalized and reached a performance level of around 90 per cent, CIGOL was not able to produce 100 per cent performance since it could not specialize the concept definition. In this paper we lay the theoretical foundations for incremental specialization techniques and describe an implementation within a new version of CIGOL.

1.2. Generalization and specialization

Niblett [12] has shown that the concept of generality can be expressed within the framework of logic. Let P and Q be two well-formed formulas. P is more general than Q if and only if $P \vdash Q$. We might also say equivalently that P is a generalization of Q or Q is a specialization of P. Note that since P and Q can be any well-formed formulas they might be atomic formulas, clauses, or conjunctions of clauses.

1.3. Previous incremental specialization techniques

Shapiro [17] describes an incremental Program Debugging System (PDS) which recognizes three different types of bug within Horn clause logic programs. Given a Horn clause program P, a ground unit goal G, and an intended interpretation M (set of ground atoms) of P, P is said to be *incomplete* when $G \in M$ and $P \nvdash G$, *incorrect when* $G \notin M$ and $P \vdash G$ and

non-terminating when $G \in M$ but G leads to a non-terminating SLD-resolution proof from P. For the purposes of this paper we are interested in the case in which PDS finds P to be *incorrect*. In the general case, when PDS finds P to be incorrect with respect to G it searches for a clause C which covers G and removes C from P. Thus let P' be the resulting program where $P = P' \wedge C$. Note that clause removal is a specialization technique since $P' \wedge C \vdash P'$. However, removing C from P is a somewhat drastic method of specializing a logic program since P' may now become incomplete with respect to a goal G' in M previously covered by C. In the 'bird' example of Section 1.1 let M be a superset of $\{Flies(Eagle), Bird(Eagle), Bird(Emu)\}$, $P = \{(Flies(x) \leftarrow Bird(x)), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$, $G = Flies(Emu)$ and $G \notin M$. As $G \notin M$ and $P \vdash G$, PDS would delete the clause $Flies(x) \leftarrow Bird(x)$ leaving $P' = \{(Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. Since P' is incomplete with respect to $G' = Flies(Eagle)$, PDS would generalize P' to $P'' = \{(Flies(Eagle) \leftarrow), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. However, whereas the goal $Flies(Sparrow)$ could be proved from $P \cup \{Bird(Sparrow)\}$ it cannot be proved from $P'' \cup \{Bird(Sparrow)\}$. In this paper we will be investigating less drastic specialization techniques than clause removal.

Wrobel [18] describes a program called MODELER which incrementally learns theories in a clausal logic formalism without function symbols. MODELER's knowledge revision module applies a heuristic approach to deal with Shapiro's incorrectness problem. MODELER saves exceptions to each rule in the form of a support set. Having found a sufficiently large set of exceptions MODELER introduces a new unary predicate to describe the exceptions. The clauses are then reformulated in terms of the new predicate. The approach used in MODELER has some similarities to that described in Section 3 of this paper. However, since MODELER, like PDS, does not use a non-monotonic logic representation it also tends to over-specialize when presented with counter-examples. In the bird example, with P, M, and G the same as above MODELER would produce $P'' = \{(Flies(x) \leftarrow Bird(x) \wedge Concept1(x)), (Concept1(Eagle) \leftarrow), (\neg Concept1(Emu) \leftarrow), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. Again, whereas $Flies(Sparrow)$ could be proved from $P \cup \{Bird(Sparrow)\}$ it cannot be proved from $P'' \cup \{Bird(Sparrow)\}$.

The approaches taken in these two machine-learning programs are typical of attempts to date to design algorithms which produce incremental specialization. However, if either of these techniques were applied to CIGOL's chess problem (Section 1.1) the performance would drop drastically from a strong performance of around 90 per cent correct to a weak, almost random performance of around 67 per cent correct. The reason is that both PDS and MODELER over-specialize. But how could one avoid over-specializing P? In Section 2 we define and investigate the properties of the most-general-specialization P' of a first-

order clausal theory P such that $P \vdash G$ and $P \nvdash G$. Although at first this seems a promising approach, we demonstrate that P' can contain an indefinitely large set of clauses. In Section 3 we show that this problem can be avoided by using a non-monotonic formalism. As an introduction to non-monotonic formalisms, in Section 1.4 we briefly describe some of the background. This approach leads to new problems in defining the notions of generality and redundancy within this formalism.

1.4. Non-monotonic formalisms

One of the most common approaches to non-monotonic reasoning is based on the 'Closed World Assumption' (CWA) inference rule. According to the CWA, if a ground atom A is not a logical consequence of a theory then infer $\overline{A}$ [5]. The CWA can be implemented in two different ways. First, by adding additional completion axioms to the theory and applying standard theorem proving techniques. This approach is exemplified by 'predicate completion' [5] and 'circumscription' [9]. Second, by employing 'Negation by Failure' (NF) where a modified theorem prover infers $\overline{A}$ whenever the attempt to prove a ground atom A finitely fails. The second approach is used within the logic programming language PROLOG for which NF has been shown to be equivalent to 'predicate completion' [5].

2. MOST-GENERAL-CORRECT-SPECIALIZATION (MGCS)

In this section we define the most-general-correct-specialization (MGCS) of an incorrect clausal theory and prove various theorems leading to a resolution-based method for constructing an MGCS. The lengthy proof of Theorem 7 from this section has been placed in Appendix A to improve readability. In the course of proving Theorem 7 we define a function called 'deriv' which turns out to be an improvement on the Kedar-Cabelli and McCarty [3] explanation-based-generalization algorithm. PROLOG code and a description of the relationship between 'deriv' and the algorithm described in [3] is given in Appendix B. First we define the correctness of first-order formulas in a weaker way than Shapiro's definition (Section 1.3). Throughout the following definitions and theorems we use the term 'intended interpretation' to mean the abstract model of an unknown formula. In any implementation we would expect to know the truth only of some ground atoms from the intended interpretation of a formula.

Definition 1 (*Correctness*) Let F be a well-formed first-order formula and M be the intended interpretation of F. We say that F is correct with respect to M, or simply F is correct, whenever M is a model of F. F is said to be incorrect otherwise.

An implementation of this definition would allow for correction of a formula with respect to known facts.

Lemma 2 (*Correct-conjunction*) Let F be the conjunction of well-formed formulas $(F_1 \wedge \ldots \wedge F_n)$ and M be the intended interpretation of F. F is correct with respect to M if and only if each F_i is correct with respect to M.
Proof Follows trivially from the fact that M is a model of $(F_1 \wedge \ldots \wedge F_n)$ if and only if M is a model of each F_i.

Definition 3 (*Correct-specialization*) Let T and T' be sets of first-order clauses and M be the intended interpretation of T. T' is said to be a correct-specialization of T if and only if T' is correct with respect to M and $T \vdash T'$.

Definition 4 (*MGCS*) Let T and T' be sets of first-order clauses and M be the intended interpretation of T such that T is incorrect with respect to M and T' is a correct-specialization of T. The set of clauses T' is said to be the most-general-correct-specialization of T if and only if $T' \vdash T''$ for every clause set T'' which is a correct-specialization of T.

This definition assumes the existence of a unique MGCS for all clausal theories. The proof of Theorem 11 provides a resolution-based method for constructing such an MGCS, which guarantees its existence. Before stating this theorem we need to prove some intermediate results.

Lemma 5 (*Subsumption by refutation*) Let A and B be two well-formed first-order formulas. $A \vdash B$ if and only if $A \wedge \overline{B} \vdash \square$.
Proof $(A \vdash B) \equiv (A \vdash B \leftarrow \blacksquare) \equiv (A \vdash \square \leftarrow \overline{B}) \equiv (A \wedge \overline{B} \vdash \square)$ by the Deduction Theorem. QED.

In the following we assume familiarity with Robinson's [16] results on resolution theorem proving. Robinson defines $R^n(T)$, for a set of clauses T, as follows

$$R^0(T) = T$$
$$R^n(T) = R^{n-1}(T) \cup \{C : C_1, C_2 \in R^{n-1}(T), C \text{ is the resolvent of } C_1 \text{ and } C_2\}$$

In addition we will define the *resolution closure* of a set of clauses as follows.

Definition 6 (*Resolution closure*) Let T be a set of clauses. The resolution closure of T, $R^*(T)$ is $(R^0(T) \cup R^1(T) \ldots)$.

The resolution closure of T does not contain all of the clauses entailed by T. The following theorem describes the relationship between the clauses entailed by T and the clauses within the resolution closure of T.

Theorem 7 (*Clause entailment using resolution*) Let T be a set of clauses and C be a non-tautological clause. $T \vdash C$ if and only if there exists D in $R^*(T)$ and substitution θ such that $D\theta \subseteq C$.
Proof See Appendix A.

Lemma 8 (*Theory entailment*) For any two sets of clauses T and T', $T \vdash T'$ if and only if $T \vdash C$ for each clause C in T'.
Proof Let T' be represented by the conjunction $(C_1 \wedge \ldots \wedge C_n)$. According to Lemma 5, $(T \vdash T') \equiv (T \wedge \overline{T'} \vdash \square) \equiv (T \wedge \overline{(C_1 \wedge \ldots \wedge C_n)} \vdash \square)$ $\equiv (T \wedge (\overline{C}_1 \vee \ldots \vee \overline{C}_n) \vdash \square) \equiv ((T \wedge \overline{C}_1) \vee \ldots \vee (T \wedge \overline{C}_n) \vdash \square)$ (2). But $((T \wedge C_i) \vdash \square) \equiv (T \vdash C_i)$ by Lemma 5. Thus (2) is true if and only if $T \vdash C$ for each clause C in T'. QED.

Lemma 9 (*Theory entailment using resolution*) Let T and T' be two sets of clauses. $T \vdash T'$ if and only if for every clause C in T' there exists a clause D in $R^*(T)$ and a substitution θ such that $D\theta \subseteq C$.
Proof Follows trivially from Lemma 8 and Theorem 7.

The following definition is similar to Plotkin's [13] definition of θ-subsumption and is used in the statement of the main theorem, Theorem 11.

Definition 10 (θ-*subsumption*) We say that clause A θ-subsumes clause B or $A \supseteq B$ if and only if there exists a substitution θ such that $A\theta \subseteq B$. Similarly, we write $A \supset B$ when $A \supseteq B$ and $B \not\supseteq A$.

Theorem 11 (*Resolution-based* MGCS *construction*) Let T be a set of clauses and M be the intended interpretation of T. Let $Q = \{C : C \in R^*(T)$ and C correct w.r.t. $M\}$ be the correct part of $R^*(T)$ and $S = \{E : E$ is a clause which is correct w.r.t. M and there exists $D \in R^*(T) - Q$ and $D \supseteq E$ and for every E', $D \supseteq E' \supset E$ only if E' incorrect w.r.t. $M\}$ be a specialization of the incorrect part of $R^*(T)$. $Q \cup S$ is the MGCS of T.
Proof From Definition 4, $Q \cup S$ is the MGCS of T if and only if $Q \cup S$ is a correct-specialization of T and $Q \cup S \vdash Q'$ for every clause set Q' which is a correct-specialization of T. From Definition 3, $Q \cup S$ is a correct-specialization of T if and only if $Q \cup S$ is correct with respect to M and $T \vdash Q \cup S$. Using Lemma 2, $Q \cup S$ is correct with respect to M by construction since each clause in $Q \cup S$ is correct with respect to M. Also, using Theorem 8, $T \vdash Q \cup S$ since T entails each clause in $Q \cup S$. Thus $Q \cup S$ is a correct-specialization of T.

We must now prove that $Q \cup S \vdash Q'$ for every clause set Q' which is a correct-specialization of T. Assume that there exists Q' which is a correct-specialization of T and $Q \cup S \not\vdash Q'$. Thus, applying Lemma 9, it is not the case that for every clause F, F is in Q' only if there exists a clause G such that G is in $R^*(Q \cup S)$ and $G \supseteq F$. That is to say that there exists a clause F such that F is in Q' and for every clause G, G is in

$R^*(Q \cup S)$ only if $G \not\supseteq F$. But since Q' is a correct specialization of T, $T \vdash Q'$. Thus, by applying Lemma 9 either there is a clause H in Q such that $H \supseteq F$ or there is a clause I in $(R^*(T) - Q)$ such that $I \supseteq F$. However, since $Q \subseteq R^*(Q \cup S)$ the first alternative contradicts the assumption letting $G = H$. Therefore we must assume the second alternative, that $I \supseteq F$. However, from the definition of S it can easily be shown that for every correct clause J for which there is a clause D in $R^*(T) - Q$ there is a clause E in S such that $E \supseteq J$. Thus either F is incorrect with respect to M, which it is not since Q' is correct w.r.t. M, or there is an E in S and $E \supseteq F$. Letting $G = E$ this contradicts the assumption and completes the proof. QED.

Theorem 11 would seem to provide the basis for an algorithm which could enumerate the elements of the MGCS of an incorrect theory T. However, this does lead to some difficulties, as the following example shows.

Example 12 We continue the 'bird' example of Section 1.3. Let T be the set of clauses $\{(Flies(x) \leftarrow Bird(x)), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$ and the true ground atoms in M, the intended interpretation of T, be a superset of $\{Flies(Eagle), Bird(Eagle), Bird(Emu)\}$, where $Flies(Emu)$ is not true in M. T is incorrect with respect to M since the clause $(Flies(x) \leftarrow Bird(x))$ is incorrect for $x = Emu$. Constructing the sets in Theorem 11 we get $Q = \{(Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow), (Flies(Eagle) \leftarrow\}$ and $S \supseteq \{Flies(Eagle) \leftarrow Bird(Eagle), (Flies(x) \leftarrow Bird(x) \wedge \neg Bird(Eagle)\}$. Note that both the clauses shown to be in S are redundant in $Q \cup S$ since they are entailed by Q. However, if we imagine that $Flightless(x)$ is true in M for all those x which cannot fly and false for all those that can fly then the clause $(Flies(x) \leftarrow Bird(x) \wedge \neg Flightless(x))$ is an element of S. But if we assume that M may contain the predicate 'Flightless' then it could contain an indefinitely large number of predicates which could be used in combination to produce an indefinitely large number of additional clauses within S.

In the next section we show that by assuming the existence of additional predicates such as 'Flightless' and by using a non-monotonic representation we can solve the problem of over-specialization of incorrect theories.

3. CLOSED WORLD SPECIALIZATION

As we stated in Section 1.4 the logic programming language PROLOG uses 'negation by failure'. In classical logic the ground literal $\overline{A}$ is provable from theory T if and only if $T \vdash \overline{A}$. In PROLOG the ground goal $notp(A)$ is provable from T if and only if A is an atom and $T \nvdash A$. In this section we

will represent clauses in the notation of Edinburgh PROLOG [2]. For example the clause $(P(x) \leftarrow Q(x) \wedge R(x))$ is written in PROLOG as

```
p(X) :— q(X), r(X).
```

But note that the PROLOG clause

```
p(X) :— q(X), notp(r(X)).
```

states that $p(X)$ is provable if $q(X)$ is provable and $r(X)$ is not provable. The literal to the left of the ':—' sign is called the 'head' of the clause, while the set of literals to the right of the ':—' sign is called the body of the clause. PROLOG proofs are carried out using SLDNF-resolution [5]. The following is an algorithm for our Closed World specialization technique.

Closed World Specialization Algorithm
Input: Set of clauses T and ground atom A such that
  $T \vdash A$ and A incorrect
Let C be the clause in T which resolved with $\overline{A}$
  in the SLDNF-resolution proof of $T \vdash A$
Let θ be the substitution for variables in C produced by
  the SLDNF-resolution proof of $T \vdash A$
If the body of C contains a literal $notp(B)$ then
  Let $T' = T \cup \{B\theta\}$
Else
  Let $\{V_1, \ldots, V_n\}$ be the domain of θ
  Let q be a predicate symbol not found in T
  Let Hd be the head of C
  Let Bd be the body of C
  Let B be $q(V_1, \ldots, V_n)$
  Let $T' = T - \{C\} \cup \{Hd{:-}(Bd \cup \text{notp}(B))\} \cup \{B\theta\}$
Output: T'

In the following example we show how this algorithm operates on the 'bird' example.

Example 13 Let T be the set of clauses $\{flies(X){:-}bird(X)),$ $bird(eagle){:-}), bird(emu){:-})\}$ and the true ground atoms in M, the intended interpretation of T, be a superset of $\{flies(eagle), bird(eagle), bird(emu)\}$, where $A = flies(emu)$ is not true in M. T is incorrect with respect to M since the clause $C = (flies(X){:-}bird(X))$ is incorrect with substitution $\theta = \{X/emu\}$. The body of C does not contain a literal $notp(B)$. $\{X\}$ is the domain of θ. Let q be 'flightless'. B is $flightless(X)$ and T' is $\{(flies(X){:-}bird(X), notp(flightless(X))), (bird(eagle){:-}),$ $(bird(emu){:-}), (flightless(emu){:-})\}$. Thus, unlike the methods in Section 1.3, $flies(sparrow)$ can be proved from $T' \cup \{bird(sparrow){:-}\}$.

An implementation in PROLOG of the Closed World Specialization

Algorithm has been completed. This version of the algorithm forms part of the CIGOL system. So far the implementation has been tested and works as expected.

4. DISCUSSION

Although, as stated in Section 1.4, non-monotonic logic is a well-explored topic, the non-monotonic representations that have been developed to date have not been applied in machine learning. From practical experience gained with the CIGOL learning system [10] it was found that it was necessary to have a specialization mechanism in order to produce monotonic performance increase. In this paper we have shown that 'minimal specializations' necessary to achieve this monotonic performance increase cannot be achieved in a classical logic representation (Section 2). They can, however, be achieved using a non-monotonic representation (Section 3). The reader should note, however, that the 'Closed World Specialization' technique described in Section 3 does not produce specialization as defined in Section 1.2 since the resultant theory T' is not entailed by the initial theory T because of the new introduced predicate symbols.

We hope also to be able to address the issue of noise within our Closed World Specialization scheme. Clearly it is important not to be too hasty in constructing new exception predicates when dealing with noisy data.

APPENDIX A—PROOF OF THEOREM 7

This appendix contains the lengthy proof of Theorem 7 from Section 2. A related result known as the 'Subsumption theorem' was proved by Lee [4]. For the purposes of the proof we will make various definitions based on the theory of resolution-theorem-proving [16]. We define a derivation expression DE as follows

> Either DE is a non-empty clause or DE is the expression $(\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle)$

where DE_1 and DE_2 are derivation expressions and l_1 and l_2 are literals found in the clauses in DE_1 and DE_2 respectively. The resolvent of a derivation expression is defined recursively as follows.

$$\text{resolvent}(DE) = \begin{cases} DE & \text{if } DE \text{ is a clause} \\ \text{Resolution of } (l_1 \vee E_1)\theta_1 \text{ and } (l_2 \vee E_2)\theta_2 \text{ resolved on } l_1\theta_1 \text{ and } l_1\theta_2 & \text{otherwise if } DE = (\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle) \text{ and resolvent } (DE_1) = (l_1 \vee E_1)\theta_1 \text{ and resolvent}(DE_2) = (l_2 \vee E_2)\theta_2 \text{ and } l_1\theta_1, l_2\theta_2 \text{ is a complementary pair} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Since the *dot*$(\cdot)$ operator represents binary resolution it is commutative, non-associative, and non-distributive. These dotted derivation expressions are an extension of notation introduced in [11]. Clearly C is an element of $R^*(T)$ if and only if there exists a derivation expression whose clauses are all elements of T and whose resolvent is C. We say that a derivation expression RE is a refutation expression whenever resolvent$(RE) = \square$. It follows from this definition of a refutation expression that RE must be a dotted expression and not simply a clause.

Next we define the partial recursive function *deriv: set of unit clauses* $\times$ *derivation expression* $\mapsto$ *derivation expression.* This function transfers the elements of the set of unit clauses found in the given refutation expression to the clause found at the root of a new derivation expression.

$$\text{deriv}(U,DE) = \begin{cases} DE & \text{if } DE \text{ is a clause and } DE \notin U \\ \text{deriv}(U,DE_2) & \text{otherwise if } DE = (\langle\{l_1\},l_1\rangle \cdot \langle DE_2,l_2\rangle) \text{ where } \{l_1\} \in U \\ \text{deriv}(U,DE_1) & \text{otherwise if } DE = (\langle DE_1,l_1\rangle \cdot \langle\{l_2\},l_2\rangle) \text{ where } \{l_2\} \in U \\ (\text{deriv}(U,DE_1) \cdot \text{deriv}(U,DE_2)) & \text{otherwise if } DE = (\langle DE_1,l_1\rangle \cdot \langle DE_2,l_2\rangle) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that deriv(U,DE) is undefined whenever DE is a clause and $DE \in U$. However, if RE is a refutation expression and $\overline{C}$ is a set of unit clauses then no sub-expression of deriv$(\overline{C},RE)$ will be undefined. A refutation expression is a derivation expression and thus, by definition, is not an empty clause. Also from the definition of deriv it will be noted that for no recursion deriv$(\overline{C},DE)$ will DE be an element of $\overline{C}$ when C is not a tautology. Clearly there is a one-to-one correspondence between the sub-expressions of the result of deriv$(\overline{C},RE)$ and a subset of the sub-expressions of RE.

Theorem 7 (*Clause entailment using resolution*) Let T be a set of clauses and C be a non-tautological clause. $T \vdash C$ if and only if there exists D in $R^*(T)$ and substitution θ such that $D\theta \subseteq C$.

Proof The 'if' part is trivial since $T \vdash D$ and $D \vdash C$, thus $T \vdash C$. We now prove the 'only if' part constructively. Let $C = (l_1 \vee l_2 \vee \ldots)$. Using Lemma 5, $T \vdash C$ if and only if $T \wedge \overline{C}$ is unsatisfiable. Let $\overline{C} = (\bar{l}_1 \wedge \bar{l}_2 \wedge \ldots)\theta_s$ where θ_s is a skolemization of the variables of C. Note that we can write θ_s^{-1} as a deterministic 'deskolemization' rewrite because of the nature of skolemization.

Following a convention introduced in [15], resolution-based refutations and derivations are drawn as binary trees. There is a one-to-one correspondence between derivation expressions and derivation trees. Figure 1 represents the transformation of a refutation tree into a derivation tree using 'deriv'. In this figure both the refutation tree and the

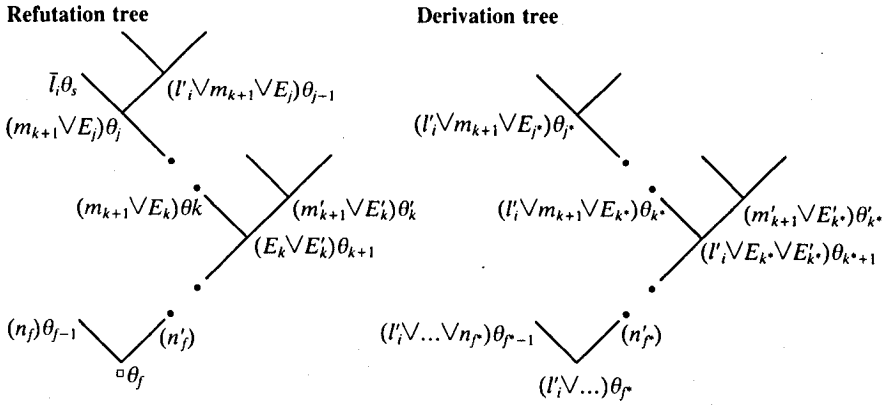


Figure 1. Transformation of refutation tree into derivation tree using 'deriv'.

derivation tree represent general cases. The diagonal ellipses (. .) merely indicate a path formed from any number of resolution steps. A capital letter or its prime with a subscript, such as E'_j, stands for a clause. A lower case letter or its prime with a subscript, such as l'_i stands for a literal found within a clause of theory T. A lower case Greek letter or its prime with a subscript, such as θ'_k represents a substitution. Note that sub-trees labelled $F\theta_h$ represent trees of maximum depth $\leq h$. The depth of a tree is defined in terms of its corresponding derivation expression as follows

$$\text{depth}(DE) = \begin{cases} 0 & \text{if } DE \text{ is a clause} \\ \max(\text{depth}(DE_1), \text{depth}(DE_2)) + 1 & \text{if } DE = (\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle) \end{cases}$$

Note also that θ_{i*} nodes in the derivation tree correspond to θ_i nodes in the refutation tree.

Let $D = (l'_i \vee \ldots)\theta_{f*}$. For the purposes of the proof it is necessary to show that there exists a most-general-unifier (MGU) for each resolution step within the derivation tree of Figure 1 and that there exists a substitution θ such that $D\theta \subseteq C$. We will first prove by induction on k^* that there exists a substitution σ_{k*} such that $\theta_{k*}\sigma_{k*} = \theta_k$ for all corresponding θ_{k*} and θ_k where $k^* \in \{0,1,\ldots,f^*\}$.

For the base case, $k^* = 0$, and thus the maximum depth of the sub-trees involved in the derivation tree is less than or equal to 0. For any such sub-tree $\theta_{k*} = \theta_0 = \emptyset$ and letting $\sigma_{k*} = \theta_k$, clearly $\theta_{k*}\sigma_{k*} = \emptyset\theta_k = \theta_k$ where θ_k is the substitution applied at the corresponding node in the refutation tree. This proves the base case. We make the inductive hypothesis that there exists σ_{j*} such that $\theta_{j*}\sigma_{j*} = \theta_j$ for all corresponding θ_{j*} and θ_j where $0 \leq j^* \leq k^*$ and $k^* > 0$. Now we must show that there exists σ_{k^*+1} such that $\theta_{k^*+1}\sigma_{k^*+1} = \theta_{k+1}$ for all corresponding θ_{k^*+1} and θ_{k+1}. With reference to Figure 1 and the inductive hypothesis we know that $m_{k+1}\theta_{k*}\sigma_{k*} = m_{k+1}\theta_k$ and $m'_{k+1}\theta'_{k*}\sigma'_{k*} = m'_{k+1}\theta'_k$. We also know from the

refutation tree in Figure 1 that $m_{k+1}\theta_k$ and $m'_{k+1}\theta'_k$ have an MGU, call it μ_{k+1}, such that $m_{k+1}\theta_k\mu_{k+1} = m'_{k+1}\theta'_k\mu_{k+1}$. Thus in the derivation tree in Figure 1 $m_{k+1}\theta_{k^*}$ and $m'_{k+1}\theta'_{k^*}$ must have an MGU, call it μ_{k^*+1}, since using the inductive hypothesis they at least have the unifier $\sigma_{k^*}\sigma'_{k^*}\mu_{k+1}$. By the definition of an MGU μ_{k^*+1} is an MGU only if there is some substitution σ_{k^*+1} such that $\mu_{k^*+1}\sigma_{k^*+1} = \sigma_{k^*}\sigma'_{k^*}\mu_{k^*+1}$. Now $\theta_{k^*+1} = \theta_{k^*}\theta'_{k^*}\mu_{k^*+1}$ and thus $\theta_{k^*+1}\sigma_{k^*+1} = \theta_{k^*}\theta'_{k^*}\mu_{k^*+1}\sigma_{k^*+1}$. Also $\theta_{k+1} = \theta_k\theta'_k\mu_{k+1} = \theta_{k^*}\sigma_{k^*}\theta'_{k^*}\sigma_{k^*}\mu_{k+1}$ by the inductive hypothesis. Since θ_{k^*} and σ_{k^*} share no variables with θ'_{k^*} and σ'_{k^*}, $\theta_{k+1} = \theta_{k^*}\theta'_{k^*}\sigma_{k^*}\sigma'_{k^*}\mu_{k+1}$. But from above $\mu_{k^*+1}\sigma_{k^*+1} = \sigma_{k^*}\sigma'_{k^*}\mu_{k+1}$ and thus $\theta_{k+1} = \theta_{k^*}\theta'_{k^*}\mu_{k^*+1}\sigma_{k^*+1} = \theta_{k^*+1}\sigma_{k^*+1}$, which proves the inductive step. Thus there is an MGU for every resolution step within the derivation. We must show now that $D\theta \subseteq C$.

Since $\overline{C} = (\bar{l}_1 \wedge \bar{l}_2 \wedge \ldots)\theta_s$ it follows that all $\bar{l}_i\theta_s$ are ground literals. Thus if $(l_i\theta_s, l'_i)$ is a complementary pair in the refutation tree with MGU γ_i, this substitution must be a ground substitution whose domain is the set of variables found in l'_i. Also, since γ_i is ground, $\gamma_i \subseteq \theta_f$ and since $l'_i\gamma_i = l_i\theta_s$ it follows that $(l'_i \vee \ldots)\theta_f \subseteq C\theta_s$. But we have proved already that $\theta_f = \theta_{f^*}\sigma_{f^*}$, and thus $(l'_i \vee \ldots)\theta_{f^*}\sigma_{f^*} \subseteq C\theta_s$. But $D = (l'_i \vee \ldots)\theta_{f^*}$ and thus $D\sigma_{f^*} \subseteq C\theta_s$. Since skolem constants can be unbound without conflict $D\sigma_{f^*}\theta_s^{-1} \subseteq C\theta_s\theta_s^{-1}$, or $D\sigma_{f^*}\theta_s^{-1} \subseteq C$. Letting $\theta = \sigma_{f^*}\theta_s^{-1}$ we complete the proof since this gives $D\theta \subseteq C$. QED.

APPENDIX B—'DERIV' AS AN EXPLANATION-BASED GENERALIZATION TECHNIQUE

The 'deriv' function of Appendix A turns out to be almost entirely isomorphic to the EBG algorithm 'prolog_ebg' described in [3]. The main differences between deriv and prolog_ebg are as follows. Firstly, while deriv acts on general clauses, prolog_ebg is restricted to Horn clauses. Secondly, prolog_ebg depends on a user-defined 'operationality criterion'. No such criterion is necessary for deriv. Thirdly, although EBG is supposed to find a 'generalization of the training example that is a sufficient concept definition for the target concept and that satisfies the operationality criterion' [3], we are not aware of any rigorous definition or proofs concerning these aims. On the other hand, it can be proved for deriv that if there is only one refutation of $T \wedge \overline{C}$ then the resolvent D of the derivation expression returned by deriv is the most general clause which both θ-subsumes C and is entailed by T. This condition places a useful bound on the complexity of D. That is that D must be smaller than C since it θ-subsumes C. No such bound exists for the size of the clause returned by prolog_ebg. In fact, when we ran prolog_ebg on the exponential function coded in PROLOG using Peano arithmetic for multiplication and addition the number of literals in the returned clause was exponentially related to the value of the exponent within the example.

This seems to indicate that prolog_ebg would not be good at helping to speed up searches in intractable domains such as game playing, though deriv might be.

We have coded a version of deriv called 'drv' which runs on Horn clause examples. The PROLOG code is given below.

```
% drv(C,D)—finds the most general clause D which is entailed by
%   the background theory and theta-subsumes C. The bodies of
%   C and D are expected to be terminated with the atom "true".

drv((H :— B),(HGen :— BGen)) :—
    functor(H,F,N), functor(HGen,F,N),
    skolemises([(H :— B)],0,_),
    drv(H,B,HGen,BGen,true).

% drv(Goal,Units,GenGoal,GenBend) — adapted Prolog
%   interpreter which proves the Goal either against program
%   clauses or against the set of skolemised Unit clauses. D's head
%   (GenGoal) and body (GenB) are formed by carrying out all
%   resolution steps except those involving Units. The atomic
%   formula "true" is passed as GenBend and placed at the end of
%   GenB.

drv(true,_,true,GenB,GenB) :— !.
drv((X,Y),U,(GenX,GenY),GenB,GenBend) :— !,
    drv(X,U,GenX,GenB,NewEnd),
    drv(Y,U,GenY,NewEnd,GenBend).
drv(G,U,GenG,GenB,GenBend) :—
    clause(GenG,GenGs),
    copy_term((GenG :— GenGs),(G :— Gs)),
    drv(Gs,U,GenGs,GenB,GenBend).
drv(Goal,U,GenGoal,(GenGoal,GenB),GenB) :—
    gmem(Goal,U).

% skolemises(Terms,N,M) — binds all variables in Terms to skolem
%   constants in the range [N..M—1]

skolemises([],N,N).
skolemises([$(N)|T],N,M) :— !,
    Np1 is N + 1,
    skolemises(T,Np1,M).
skolemises([Tm|T],N,M) :—
    Tm =.. [_|Tms],
    skolemises(Tms,N,O),
    skolemises(T,O,M), !.
```

```
% gmem(Goal,Units) — unifies Goal to a member of Units if
%   possible. Otherwise fails.

gmem(X,X).
gmem(X,(A,B)) :-
    gmem(X,A) ; gmem(X,B).
```

This was run on the 'suicide' example from [3]. The background knowledge was

```
kill(A,B) :— hate(A,B), possess(A,C), weapon(C).
hate(W,W) :— depressed(W).
possess(U,V) :— buy(U,V).
weapon(Z) :— gun(Z).
```

When given the goal *drv*((*kill*(*john,john*), :— *depressed*(*john*), *buy*(*john,gun1*), *gun*(*gun1*), *day*(*tuesday*), *true*), *D*), *D* is instantiated to (*kill*(*X,X*) :— *depressed*(*X*), buy(*X,C*), *gun*(*C*), *true*). In this case drv produces the same result as prolog_ebg. However, given the standard recursive definition of list membership, the goal *drv*((*member*(*3,*[*1,2,3*]) :— *true*), *D*) instantiates *D* *to* (*member*(*A,*[*B,C,A*|*D*]) :— *true*). On the same problem prolog_ebg merely produces the standard recursive clause (*member*(*A,*[*B*|*C*]) :— *member*(*A*,*C*)), which is not a useful generalization.

REFERENCES

1. Chang, C. and Lee, R. (1973). *Symbolic logic and mechanical theorem proving.* Academic Press, London.
2. Clocksin, W. F. and Mellish, C. S. (1981). *Programming in Prolog.* Springer-Verlag, Berlin.
3. Kedar-Cabelli, S. T. and McCarty, L. T. (1987). Explanation-based generalization as resolution theorem proving. In *Proceedings of the Fourth International Workshop on Machine Learning,* (ed. P. Langley) pp. 383–89, Morgan Kaufmann, Los Altos.
4. Lee, C. (1967). *A completeness theorem and a computer program for finding theorems derivable from given axioms.* PhD thesis, University of California, Berkeley.
5. Lloyd, J. W. (1984). *Foundations of logic programming.* Springer-Verlag, Berlin.
6. Michalski, R. and Larson, J. (1978). *Selections of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11.* UIUCDCS-R 78-867, Computer Science Department, Univ. of Illinois at Urbana-Champaign.
7. Michalski, R. S. (1983). A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach* (eds R. Michalski, J. Carbonnel, and T. Mitchell) pp. 83–134, Tioga, Palo Alto, CA.
8. Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence,* **18**, 203–26.
9. Genesereth, M. R. and Nilsson, N. J. (1987). *Logical foundations of artificial intelligence.* Morgan Kaufmann, Los Altos.

10. Muggleton, S. H., Bain, M. E., Hayes-Michie, J. and Michie, D. (1989). An experimental comparison of human and machine learning formalisms. In *Proceedings of the Sixth International Workshop on Machine Learning*, Kaufmann.
11. Muggleton, S. H. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–52, Kaufmann.
12. Niblett, T. (1988). A study of generalization in logic programs. In *EWSL-88*, Pitman, London.
13. Plotkin, G. D. (1971). *Automatic methods of inductive inference*. PhD thesis, Edinburgh University.
14. Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In *Machine Learning: An Artificial Intelligence Approach*, (eds R. Michalski, J. Carbonnel, and T. Mitchell), Tioga, Palo Alto, CA.
15. Robinson, J. A. (1965). Automatic deduction with hyper-resolution. *Intern. Jour. of Computer Math.*, **1** pp. 227–34.
16. Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *JACM*, **12** No 1 pp. 23–41.
17. Shapiro, E. Y. (1983) *Algorithmic program debugging*. MIT Press.
18. Wrobel, S. (1988). Automatic representation adjustment in an observational discovery system. In *EWSL-88*, pp. 253–62, Pitman, London.

9

Interactive Induction

W. Buntine†
University of Technology, Sydney
Australia

D. Stirling
BHP Steel International Group,
Australia

Abstract

A variety of artificial intelligence induction systems are being successfully used for knowledge acquisition. In some of these systems, knowledge acquisition is not an automatic process but rather an *interactive* process between expert/user and system, with induction serving as the focus. We refer to this generic approach as interactive induction. In this paper we examine some case studies, and discuss the approach and software facilities required to support it. Key features for success are the methods and man-machine environment employed to elicit prior, subjective information from an expert (in order to augment the knowledge implicit in the training data), together with the expert's acceptance and validation of the knowledge induced.

1. INTRODUCTION

A common task in building knowledge-based systems is inducing a rule to handle classification. Given a set of classified examples of a concept, the task is to develop a classification rule to predict the class of further unclassified examples. This task is performed by induction systems as an aid to knowledge acquisition [23, 13] when examples are available but more general, explicit knowledge is insufficient.

Artificial intelligence induction systems are successful because they match two key requirements for expert systems: validation and user/expert acceptance. Not only do they produce knowledge, essentially from fragments of knowledge, but they also pay careful attention to the task of gaining the expert's acceptance (and validation) of the knowledge produced. Two features of knowledge that are important for this are *accuracy* and *comprehensibility*.

Buntine [5] showed that when these systems are applied to training data

†Present address: Turing Institute, Glasgow, UK.

representing a historical data base, their inductive power (that is, the potential for inducing accurate knowledge) must come from additional *subjective information* provided by the expert. The term 'subjective' applies because the information often does not come directly from data or other objective sources but from the expert's own experiences.

An expert can provide this subjective information in several ways using current commercially available induction tools [14]. The expert may provide 'useful' attributes (features or descriptors) for describing examples. This provision does not necessarily have to be made overtly. With medical records, for instance, only relevant decision-aiding details about a patient tend to be recorded. It is these details that are passed to the induction system, along with the expert's implicit stamp of approval. Secondly, the expert may appraise results of induction and possibly resubmit certain fragments for further processing. Finally, the expert may submit a set of hand crafted tutorial examples about which knowledge is expected to be built. Recent developments extend this list considerably [1, 3, 12], providing post-editing of rules, interactive building of rules, manipulation of data (for instance, user-defined thresholds for numeric attributes), and several other features for expert interaction.

There are, of course, other methods for obtaining subjective information from an expert: the traditional manual knowledge acquisition method [8], and hybrid methods such as computer-aided interviewing [4] and computer-aided knowledge refinement [20]. Although sophisticated aids for input, interviewing, and editing are improving the speed of these interview-driven methods, they are still time-intensive for the expert. In addition the knowledge is not always reliable (examples of this will be given in Section 2). Significant inconsistencies between what an expert says he does, what he actually does, and what he should have done by hindsight are common [8]. For instance, in the context of uncertainty, people have limitations with reasoning and in articulating their reasoning [10], so knowledge elicited must be interpreted with caution [7]. This can be partially overcome by working in a structured probabalistic environment [9, 26], although maybe at the expense of even slower development time.

In this paper we identify and discuss a generic induction approach we call *interactive induction* that offers an alternative method for eliciting subjective information from an expert during the course of knowledge acquisition. The approach extends 'pure' induction by involving the expert in the provision of additional subjective knowledge and in the incremental evaluation and validation of the knowledge induced. It is economical with the expert's time because it performs these tasks in a way that the expert feels comfortable with. Like other induction approaches and in contrast with non-inductive knowledge acquisition ones, it is still able to induce knowledge beyond that which is articulable

or known by the expert. Unlike earlier induction methods, induction is not viewed as an automatic process, and interaction does play a key role.

We present a number of case studies in Section 2, and in Section 3 consider aspects of the knowledge engineering problem pertinent to the interactive induction approach. In Section 4 we discuss software features needed to support this approach.

2. CASE STUDIES

In this section we introduce and discuss some induction cases that illustrate various styles of interactive induction across different domains and induction tasks.

An interactive approach to induction was first reported by Shapiro [27]. He extended an ID3-like rule induction algorithm so that the tree-building process itself, the selection of tests at each new decision tree node, could be guided by the user/expert. Michie [15] describes an application of this technique to the credit assessment domain, in which a 5 per cent increase in accuracy over automatically generated trees was obtained.

Kodratoff and Tecuci describe the DISCIPLE system [11], a system that interactively develops rules from data by extracting explanations from an expert. Expert-supplied explanations, like interview-obtained rules, are well known to be often incomplete or overly general. Nevertheless, explanations are a common source of subjective information in the training of novices; it should come as no surprise that they can also play a key role in inductive knowledge acquisition.

The remaining subsections illustrate different facets of interactive induction: knowledge-base bootstrapping, the use of an oracle during induction, and the role of explanation for validation.

2.1. Producing coated steel

This first case concerns the production of coated steel products [28]. These products have different steel type, formability, surface finish, length, etc, and a plant should be able to produce thousands of such products. In addition, potentially thousands of different processing sequences can be used in their manufacture. For instance, there are various machines for annealing, reducing, slitting, shearing, painting, packing, etc.; each machine can only operate on certain kinds of steel and needs to be used in the correct sequence.

Our problem† was to develop a system to suggest *routings* (a sequencing of machines along with machine instructions) for the manufacture of

†This reports exploratory research being undertaken jointly between the Sydney Expert Systems Group and BHP Steel International Group, Coated Products Division (CPD).

new products. The expert† for this task helped design the original software used for controlling CPD's Port Kembla facility and is familiar with the possible kinds of steel products, existing routings, the machinery used, and general metallurgy.

As to the problem at hand, the expert could, after some discussion, suggest a general strategy, provide worked-through examples, general hints, and a few rules of thumb, and so on.

As a result, we broke the problem up into two parts and chose to focus initially on 'cold-rolled' steel products. The process grammar problem described in Section 2.1.1 induces a simple grammar to determine which routings can be used for these products. The processing unit selection problem described in Section 2.1.2 finds, for each branch in the grammar, which path should be taken by a particular product. For instance, of all the 'cold reduction' machines (as found in a particular place in the grammar), which should be used for the manufacture of CA50T (cold rolled, aluminium killed, temper-rolled steel with a minimum hardness of 50 Rockwell B)?

Even though the expert was skilled at the problems concerned, after the first few interviews it was clear that manual knowledge acquisition would be a protracted process. His difficulty was, as usual, with articulation: expressing knowledge at the right level of abstraction and degree of precision, organizing knowledge and ensuring consistency and completeness. For instance, he sometimes gave merely a list of special cases, or an over-generalization and only thought of the exceptions when they were actually raised later.

Fortunately, there were several thousand examples available in the existing data base of products and routings, but taking into account the number of decisions needed to arrive at a particular routing, this amount of data is not really significant (see Section 4 in [5]). Additional, subjective information was needed. Preliminary trials with induction indicated a second problem existed: essentially, the expert knew too much. He had a partial idea of what was and what was not appropriate knowledge and he was, invariably, not satisfied with some of the output of an induction system.

So we needed to tap into the expert's subjective information, even if it was not entirely accurate, without going through a protracted series of interviews, and we needed to gain his acceptance of the knowledge produced. In the remainder of this section, we describe our experiences on the problem to date. Although the prototype is still under development, the knowledge acquisition and validation phase is mature.

†John Wiltshire of CPD.

2.1.1. *Process grammars*

This subsection introduces the approach taken with the overall problem: *knowledge-base bootstrapping.* We describe how a process grammar was developed for the cold-rolled products.

In an initial interview, we asked the expert to 'describe the kinds of routings usually found for cold-rolled products'. His final answer is illustrated in Figure 1. This confirmed two details essential for the bootstrapping approach: a reasonable grammar should exist, and the expert should be sufficiently articulate to reason about general statements on routings. This initial interview also provided information useful for the induction process. For instance, 'transfer' steps could be deduced from other parts of the routing, the location of the machines used before and after a transfer, and 'test' steps were oddities that could be inserted later.

pickle-line
five-stand-mill / reversing-mill
coil-temper-mill / coil-anneal / cont.-galv.-line
[paint-line / tension-leveller(TL)]
pack(generic)
dispatch(generic)

Figure 1. Expert's initial grammar. (Bracketed ([ ]) items are optional. Items on the same line, separated by '/' are alternatives. Items otherwise follow each other sequentially.)

We then induced a grammar from the existing set of cold-rolled routings (without using the expert's initial grammar). This induced grammar was a sufficient starting point for the expert. It forced him to think of all the exceptions and gave him a recognizable grammar to work with. He was sufficiently unfamiliar with grammars, flow charts, and so on, to make this assistance valuable.

The expert made a number of refinements to the grammar and provided names for many of the stages. A simplified version of the final grammar is shown in Figure 2. In comparison with Figure 1, notice the extra detail and the change in the overall structure (reversing-mill appears in two places and paint-line and tension leveller can occur together in the one routing).

With little effort by the expert, we were able to develop a grammar for cold-rolled products. This grammar, validated by the expert and on test data, now provides the framework about which the remainder of the system is being developed. Induction in this knowledge acquisition episode has served the purpose of bootstrapping subsequent manual knowledge refinement. This allowed a more complete grammar to be developed and validated than through either induction or interview alone.

pickle:[pickle-line [test]]
reduction:[five-stand-mill [test] / reversing-mill [test]]
annealing:[decarb.[test] / coiler open-coil-anneal coiler /
[clean]coil-anneal / cont.-galv.-line / clean]
temper:[coil-temper-mill [coil-temper-mill/test] /
reversing-mill [test]
tension:[EGL TL / TL EGL / EGL[test] / TL[test]]
finishing-stages:[paint-line(generic)]
[slit][shear[reshear]] / [elec.-steel-slit][shear] / Yoderline
pack(generic)
dispatch(generic)

Figure 2. Induced grammar after expert's refinements. (Stages are in italics.)

2.1.2. *Processing unit selection*

When applied in the role of knowledge-base bootstrapping, induction systems need more than just accurate rules output. This subsection demonstrates the importance of the induction user-interface to the process of knowledge-base bootstrapping. We describe how rules were developed for selecting particular processing units. For instance, in the annealing stage, which of the six options should be chosen for a given product? This kind of knowledge would complement the process grammar so that routings could be developed.

We asked the expert, 'for each stage of the process routing how might you chose between several alternatives?' This was reasonable to ask as he had identified that the stages were meaningful to him. He responded by formalizing various example rules as illustrated by part of the pickle stage set shown in Figure 3a. Using ID3 [21] and the rich set of attributes in the product data base, decision trees were grown for each multiple path stage in the grammar of Figure 2. The decision tree and a new rule for the pickle stage are also given in Figure 3. The expert's initial attempt at formalizing his knowledge brought out a combination of specific and fuzzy attributes such as, 'if the product is CK1055 and it is overly thick then don't pickle'. In using ID3 we were able both to identify missing or yet-to-be formalized knowledge and adequately to quantify previously vague attributes and their values. Most of the induced decision trees formed reliable generalizations of the specific instances cited by the expert.

The decision trees generally gave high accuracy on a test set, but they were not acceptable to the expert. Some of the problems could have been recognized and accounted for before the induction process began by using an appropriate induction methodology (to our knowledge, no general methodology for the interactive style of induction has yet been reported). However, in general, we believe the expert needs to be

Default is pickle only:

rule 1: If steel = C & thickness > 3.2 mm
then don't pickle
rule 2: If product (CA70T or CA60T or CK1055) & overly thick
then don't pickle
rule 3: If product (CM350-G)
then pickle and test

(a) Rules for pickle stage suggested by expert.

steel = —: [pickle-line] (0)
steel = R: [pickle-line] (62)
steel = A: [pickle-line] (102)
steel = U: [pickle-line] (33)
steel = S: [pickle-line] (8)
steel = W: [pickle-line] (4)
steel = V: [pickle-line] (7)
steel = K:
— thick ≤ 2.25: [pickle-line] (15)
— thick > 2.25: [don't pickle] (10/2.0)
steel = M:
— strength > 475: [pickle-line] (14)
— strength ≤ 475:
— — thick ≤ 1.3995: [pickle-line] (5/1.0)
— — thick > 1.3995:
— — — width ≤ 609.949951: [pickle-line] (3)
— — — width > 609.949951: [don't pickle]

(b) Decision tree built for stage.

rule 4: If steel = K & thickness > 2.25
then don't pickle

(c) Additional rule from decision tree omitted by expert.

Figure 3. Combining induced and elicited knowledge.

prompted with incomplete rules before the relevant knowledge will spring to mind. As experience with induction increases, need for prompting decreases.

For instance, a test was often made on 'length' in decision trees on early pre-painting and shearing portions of the routing grammar, when

in fact length was known by the expert only to be important in the later stages. All steel products are initially treated as continuous coil. Width or thickness are much more relevant attributes in early processing. A similar situation existed for several other attributes. When inducing rules for certain stages of the grammar, the use of such irrelevant attributes needs to be suppressed.

Also, the 'length' attribute can only be meaningfully tested after another attribute, 'shape', has been evaluated. Only products of shape 'sheet' have a 'length'. Other attributes encountered also exhibit a similar dependency. Several of the decision trees that tested 'length' before 'shape' look ridiculous to the expert. To overcome this kind of problem, an induction system needs to be able to pre-define a precedence on attributes, or more general controls on allowable classes of rules. A similar problem exists in the credit-assessment domain (that is, credit card applications) where managers are reluctant to accept rules that, regardless of measured performance, have some obvious flaws [6].

Finally, the treatment by ID3 of continuous variables was sometimes confusing to the expert. These subsequently needed to be readjusted to the nearest recognizable break-point to assist the interpretation of the decision tree concerned. Further discussion with the expert revealed that the process routing domain contains a reasonably static set of values for some continuous variables. For example, typical thickness break-points are: <0.3 mm, <0.7 mm, $\leqslant 1.6$ mm and >1.6 mm. It is clear that an induction system needs to front-end onto a data base system so that flexibility is available in the choice of attributes as the induction cycle progresses.

As well as being processed by ID3, the data were later processed by Quinlan's C4 [22, 23] to provide a comparison. This is an enhanced form of the ID3 induction tool that generates explicit conjuctive rules instead of decision trees. An example of C4 output appears in Figure 4.

These rules were much more comprehensible to the expert than straight decision trees. Sets of conjunctive rules are, in general, simpler and more comprehensible than trees. Comprehensibility is essential in the interactive approach. C4 also gave a measure of strength for each rule—a useful guide to whether the rule could be ignored or whether it needed further thought on the expert's part. We were able to get considerably more feedback from the expert when working with C4 rather than ID3. For instance, of the rules initially produced by C4, about a half were acceptable to the expert (note that despite only fair acceptance, these rules had a high accuracy). Some examples are given in Figure 4a. Poor choices for tests, however, were found in other rules. An example is shown in Figure 4b. While such a rule is not too inaccurate, it exhibits coincidental attribute and class relationships that, to the expert, are not acceptable knowledge for designing routings.

Default class is tight coil annealed

Rule 1:
family = TN (0.08)
⇒ continuous galv. line [99.4%]
Rule 2:
steel = R (0.73)
condition = S (0.82)
surface-quality = - (0.28)
⇒ IHI coiler, OCA, IHI coiler [99.3%]
Rule 3:
surface-quality = D (0.05)
⇒ cleaning line & coil anneal [99.1%]
Rule 5:
steel = V (0.04)
thick > 0.75 (0.85)
thick < 2.5 (0.92)
⇒ IHI coiler, OCA, IHI coiler [94.4%]
Rule 16:
hardness > 82.5 (0.03)
⇒ no annealing [98.4%]
Rule 23:
steel = M (0.40)
strength > 450 (0.54)
⇒ IHI coiler, OCA, IHI coiler [96.3%]

(a) Good and robust.

Rule 15:
hardness > 75 (0.03)
width > 39.5 (0.93)
⇒ no annealing [98.5%]

(b) Weak with unrealistic dependency on width.

Figure 4. Example of induced rules for annealing stage rated by domain expert.

In addition, the expert was willing to modify rules and to suggest further changes for future induction on the same data. Two examples are shown in Figure 5. After reviewing the reduction stage rules he suggested that the edge trimming attribute 'bw/me' and others should be suppressed in induction in the basic stages. He knew these attributes to be relevant only in the finishing processes. Also, an annealing stage rule needed to be modified to reflect the maximum width constraint on the

Rule 14:

bw/me = B (0.20) ... removed attribute from induction
thick < 1.8 (0.94)
width > 1320.0 (0.99)
⇒ five stand mill only [99.3%]

(a) Identification of inappropriate attribute for stage.

Rule 26:

strength < 450 (0.23)
width > 805 (0.08)
width < 1137.5 (0.10) ... modified to 1075 mm
group = S (0.03)
⇒ decarb. and test [80.0%]

(b) Limitation on width processing for unit.

Figure 5. Example of expert's modification of rules.

decarburizing unit. Notice that in the first case involving the inappropriate attribute, C4's measure of the importance of the attribute is low.

After considering the work so far, the expert was able to develop a more refined notion of which attributes were relevant for stages in the processing. Using the reduced set of attributes for the basic processing stages, pickle to painting (in Figure 2), another iteration of rule induction with C4 was made. This improved the rule acceptance level by some 5–10 per cent to a rate of about 50–60 per cent on average. This is not as high as was hoped; the expert believes there are some atypical groups of routings in the product data.

Knowledge acquisition, after some initial interviews, focused around several iterations of induction. The expert's interaction contributed in two ways. First, he helped in constraining the induction input, in reframing each induction problem, and in correcting data so that more effective and comprehensible rules could be produced. This seemed to be an evolutionary process, evolving as the expert became more aware and articulate about the kinds of knowledge needed to improve the induction system's performance. Secondly, the expert acted as an oracle, in refining and validating induced knowledge. Rules that, perhaps after modification, were accepted by him could then be put aside for inclusion in the final knowledge-base. Both these modes of interaction, reframing the induction problem and refining the induced knowledge, seemed to us to be essential when working with a particularly articulate expert. Throughout this interaction the expert's own knowledge has also been improved and formalized. For instance, a clean-up of the existing

routings data base is now underway, based on the expert's experiences during knowledge acquisition.

2.2. New concepts in neuro-psychology

This case describes some work by Muggleton [18, 17] that demonstrates the importance of involving the expert, as an oracle, in the process of vetoing and validating new concepts.

With several days' work for a domain expert, a system was inductively developed for deciding dysfunction of the left parietal brain area. This system performed considerably better than an existing manually-built expert system and a second generation manually-built system several months into its development cycle (90 per cent compared with 82 per cent and 65 per cent agreement with expert). The knowledge produced was also more comprehensible to the expert and consequently gave better explanations when operating.

The general approach used was, in our terminology, interactive induction. The Duce system [19] suggested new concepts potentially useful for deciding dysfunction, and concurrently induced rules. The expert acted as an oracle for concepts; that is, the expert's role was to veto concepts suggested by the system and to name appropriately those that were accepted. This provided needed subjective information and validated the worth of the intermediate concepts at the same time. The importance of the interaction was confirmed experimentally. Without the expert's involvement the induction system performed poorly at the same task.

Potential concepts were presented to the expert in a manner with which she felt comfortable; this was critical to the system's performance. In an early version of Duce, concepts were presented in rule form. Because examples were highly abstracted and reduced case records, this representation was particularly foreign to the expert and so, typically, incomprehensible. In the current Duce, rules are presented by means of examples instead. The veto process was also speeded up by allowing the expert an option to specialize a concept if it was not appropriate. This helped guide the search for appropriate concepts.

2.3. Diagnosing electrical transformers

This case describes some work by Muggleton and Riese [16, 24] that highlights the need for comprehensibility, demonstrating the importance of the 'explanation' in the process of validating induced rules.

A system called EARL was developed for diagnosing imminent breakdown in large oil-cooled electrical transformers. The service interruption and repair or replacement cost of these can run into millions of dollars; the accepted rate of human diagnostic failure in this domain is below 0.1 per cent.

When the final (inductively built) system was validated against 859 test cases, it was found that the system was in total agreement with the expert as far as the diagnosis was concerned. However, in four of the 208 cases in which the expert and system agreed a problem existed, closer inspection revealed that the underlying reasoning of the expert and system differed. These cases were interpreted as erroneous by the expert and subsequent refinement was needed. Without comprehensibility of induced knowledge, these problem cases could not have been located.

3. KNOWLEDGE ENGINEERING ASPECTS

With the constraints of minimizing the expert's time, and producing knowledge with an acceptable level of accuracy and comprehensibility to aid the validation and evaluation tasks, knowledge acquisition has all the ingredients of an intriguing optimization problem in a man–machine setting. In this section we make an overview of features and tasks in the knowledge engineering process pertinent to interactive induction.

Whether induction, interactive induction, or interview is appropriate for knowledge acquisition depends on a range of issues; some are given in Table 1. The main choices to be made are on the method of validation and on the particular style of acquisition.

Not surprisingly, the *validation* of induced knowledge has become a key task assisted by the interactive approach. Factors affecting the choice of validation methods are essentially those in Table 1. For instance, if a sufficiently large amount of data is available accuracy can be reliably determined merely by performance of the system on that data. Typically this is not the case, so another standard method of software validation is usually adopted: perusal of the 'code' by someone

Table 1. Features of a knowledge acquisition problem.

Feature	Description	Section
Expert's capabilities	How articulate and reliable is the expert? What sort of subjective information can he conveniently provide?	2.1
Amount of data	Less data needs to be complemented with more subjective information	2.1
Domain features	Amount of prior knowledge available, noise, etc.	2.1.2, 2.2
Validation requirements	Degree of accuracy, confidence, and comprehensibility required	2.3
Decision requirements	How complex (for instance, in bits) is the decision? Yes/no, real valued, etc.	

sufficiently expert in the domain and software specification/requirements. In the knowledge acquisition context, code corresponds to induced knowledge and usually only the expert is qualified sufficiently to perform the perusal. For this reason, adequate presentation of rules has also become an important task for induction systems.

Three contrasting approaches to validation were given in the case studies.

The validation of the EARL system (Section 2.3) used an approach combining data and expert validation. On this application, a very high accuracy was required of the final product so extra care was needed in validation. The system was first compared with test cases; on certain cases the reasoning of the system was compared with that of the expert; on a few cases, the diagnoses were actually confirmed by an (expensive) full analysis of the actual transformers.

Duce (Section 2.2) performs validation interactively. New concepts are validated by an oracle (the expert) right after they are proposed. The general technique of incremental, interactive validation was also employed in Sammut and Banerji's MARVIN system [25].

In the coated steel application (Section 2.1), the process of validation was tied to the rule refinement process. In this particular application, the expert was sufficiently skilled and articulate about the domain so that knowledge refinement by him was both feasible and acceptable from the point of view of validation.

A summary of validation and other tasks for the interactive induction approach are given in Table 2.

Although the choice of knowledge representation in induction systems is usually fixed, the choice of language (that is, attributes, etc., for describing examples) and data is up to the user. The *problem formulation* task is about the making of these choices, and the evolving task of refining them. The coated steel application demonstrates that an expert can perform reformulation successfully as his understanding of induction improves and his knowledge of the domain becomes clearer. Westinghouse Electric's nuclear fuel division [2] describes an application where careful choice of examples proved successful. In this application classification was not binary or discrete (for instance, yes/no), but rather on a continuous scale from good through average to poor. A decision tree induction system was not strictly appropriate. By supplying only very good and very poor examples to the decision tree induction system it was able to produce results.

The *rule presentation* task is about presenting rules in a manner aiding their comprehension by the expert. For instance, in the coated steel application, conjunctive rules were preferred by the expert over decision trees when he performed validation or refinement. Keeping rules simple and tests in rules meaningful, however, is not always a sufficient means of

Table 2. Interactive induction tasks.

Task	Description	Section
Problem (re-)formulation	Setting-up appropriate language for describing examples (formulating relevant attributes), obtaining examples	2.1.2
Incorporation of prior knowledge	Obtaining and using subjective information in the form of partial rules, preferences, etc., or validated rules cycled back	
Interactive rule building	Having expert/user guide rule building process interactively, see also interactive validation	
Interactive validation	Involve the expert in validation of key, generated examples and rules during induction	2.2
Rule presentation	Presenting induction results in a manner aiding comprehension and validation	2.1.2, 2.2
Final validation	Validation after induction by test set or perusal by expert	2.3
Rule refinement	Facilities and suggestions for post-editing of rules	2.1.1, 2.1.2
Validation of data	For instance, assist expert by flagging outliers as possible bad data	

presentation for comprehension by the expert. In the neuro-psychological application (Section 2.2) it was found through experience that new concepts needed to be explained by means of examples rather than in abstract form.

Quinlan, Compton, Horn, and Lazarus [23] describe an example of the *data validation* task. In the training set used there were 10 examples that were outliers according to the induced decision tree. These were referred back for expert review. Most were found to have been incorrectly extracted from the original narrative records.

Finally, appropriate man–machine interaction is essential for success with the interactive approach. It is the role of the system to facilitate communication with the expert, to try and extract from him the kinds of knowledge he can conveniently provide. Table 2 can also be viewed as a tabulation of interaction modes between expert and induction system. What information can actually be provided by the expert depends on his articulation skills with respect to the problem at hand and the nature of his cognitive grip on the domain. These, in turn, depend on the expert's experience with teaching or explaining his methods and whether his skill is performance-oriented or developed to a more abstract level.

The neuro-psychological application illustrates some of these principles of communication. In this noisy domain, the expert was able to

identify concepts critical to the decision-making process, but is not able to articulate rules; Duce is able to suggest potentially useful concepts and then incorporate these in induced rules. Duce and the expert interact accordingly to achieve their common goal.

4. SOFTWARE FACILITIES

Commercial induction tools provide facilities for the rules refinement task and for the manipulation of examples, for instance, redefining attributes. These facilities can readily be implemented using existing technology. In accord with good software practice, however, induction systems are being integrated with existing information systems tools such as data base and knowledge-base management systems (with browsers, interpreters, etc.). Many of these facilities then come free.

Other tasks described in the previous section are not so peripheral to induction proper and require facilities built into the central induction system, often from the initial design stage. We shall briefly mention three types that we believe should be the focus of more intensive study.

C4 [22] produces rough statistics on the applicability of and confidence in a rule and each of its components. Many variations on this theme are possible.

Duce [19] has the expert act as an oracle. The expert has to answer questions posed by the system, to veto and validate potentially useful concepts. There are other ways an induction system can use an oracle. For instance, explanation could be obtained from the oracle instead of merely confirmation [11]. Also, the system could generate critical examples of known concepts to help differentiate competing hypotheses.

Finally, typical decision tree induction systems currently allow no prior knowledge to be used in induction other than the default preference for simple rules [5]. (This default preference is a consequence of the choice of 'relevant' attributes.) Subjective information should be taken advantage of where possible, and appropriate facilities provided. In the coated steel application, the expert could give prior information concerning the degree of relevance of some attributes. During knowledge acquisition for influence diagram systems [26], experts routinely convey information about independence between some (sets of) attributes. This information allows an induction problem to be factored into smaller subproblems.

5. CONCLUSION

It was argued that induction for the purposes of knowledge acquisition should be interactive so that further subjective information can be input to the process and so that the final induction product can gain the

expert's acceptance. A number of cases have been reported which demonstrate that the generic interactive induction approach gives superior performance in real knowledge acquisition tasks to non-interactive induction and to knowledge acquisition by interview ('dialogue acquisition method').

A special case of interactive induction called knowledge-base bootstrapping has been discussed. This uses induction to draft a knowledge base for subsequent refinement by the skilled and articulate expert.

The interactive induction approach has been viewed along a number of lines: which features are relevant to the choice of knowledge acquisition style, which tasks should be performed by expert and system during the course of induction, and which kinds of subjective information the expert can conveniently provide.

Finally, a number of areas that have not been fully explored here, but seem critical to the generic approach, are the generation of statistics concerning the applicability and confidence in a rule, the use of an oracle for interactive validation and induction guidance, other modes of man–machine interaction, and interactive induction methodologies.

Acknowledgements

This paper was drafted while the first author was studying at the Turing Institute, Glasgow. We are grateful to BHP Steel International Group, Coated Products Division for permission to publish details of the joint project and to Ross Quinlan for the use of C4. We are also grateful to Steven Muggleton, Donald Michie, and others at the Turing Institute for their insights into the pragmatics of inductive knowledge acquisition and the Turing Institute for supporting the first author's visit to Turing and attendance at the MI-12 Workshop, in Tallinn, Estonia, USSR in 1987.

REFERENCES

1. A-Razzak, M. (1987). *Dealing with noisy data using the CX algorithm: User documentation.* Technical Report, Intelligent Terminals Ltd., Glasgow.
2. Westinghouse Electric's Nuclear Fuel Division (1985). A letter. *Expert Systems*, Jan. p. 20.
3. Al-Attar, A. Personal communication. Attar Software Ltd., Leigh, Lancaster, UK.
4. Boose, J. H. (1986). *Expertise transfer for expert system design.* Elsevier.
5. Buntine, W. L. (1987). Decision tree induction systems: a Bayesian analysis. In *Third Workshop on Uncertainty in Artificial Intelligence*, AAAI, Seattle, Washington.
6. Carter, C. and Catlett, J. (1987). Assessing credit card applications using machine learning. *IEEE Expert*, **2** No. 3, 71–9.
7. Cleaves, D. A. (1986). Cognitive biases and corrective techniques: proposals for improving elicitation procedures for knowledge-based systems. In *Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada. *International Journal for Man-Machine Studies*, **27** No. 2, 155–66.
8. Hayes-Roth, F., Waterman, D. A. and Lenat, D. (eds) (1983). *Building expert systems.* Addison-Wesley.
9. Henrion, M. and Cooley, D. R. (1987). An experimental comparison of knowledge

engineering for expert systems and for decision analysis. In *Sixth National Conference on Artificial Intelligence*, pp. 471–6, Seattle.
10. Kahneman, D., Slovic, P. and Tversky, A. (1982). *Judgement under uncertainty: heuristics and biases.* Cambridge: Cambridge University Press.
11. Kodratoff, Y. and Tecuci, G. (1987). DISCIPLE: an integrated expert and learning system for weak theory domains. Unpublished report.
12. Hassan, T. Personal communication. Intelligent Terminals Ltd., Glasgow, UK.
13. Michalski, R. and Chilausky, R. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean diagnosis. *Policy Analysis and Information Systems*, **4** No. 2, 125–60.
14. Michie, D. (1986). The superarticulacy phenomenon in the context of software manufacture. *Proc. Roy. Soc.* (*A*) **405**, 185–212.
15. Michie, D. Personal communication, commenting on a case described in *Problems of computer-aided concept formation.* In *Applications of expert systems 2* (ed. J. R. Quinlan) pp. 310–33. Addison-Wesley.
16. Muggleton, S. (1990). *Inductive acquisition of expert knowledge.* Turing Institute Press in association with Addison-Wesley.
17. Muggleton, S. *Inverting the resolution principle.* This volume.
18. Muggleton, S. (1987). Report on savant task 31: investigation of machine learning for the neuropsychological expert system. Unpublished report.
19. Muggleton, S. (1987). Structuring knowledge by asking questions. In *International Joint Conference on Artificial Intelligence*, pp. 287–92, Milan.
20. Politakis, P. G. (1985). *Empirical analysis for expert systems.* Pitman, Boston.
21. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, **1**, No. 1, 81–106.
22. Quinlan, J. R. (1987). Generating production rules from decision trees. In *International Joint Conference on Artificial Intelligence*, pp. 304–7, Milan.
23. Quinlan, J. R., Compton, P. J., Horn, K. A. and Lazarus, L. (1987). Inductive knowledge acquisition: a case study. In *Applications of expert systems*, (ed. J. R. Quinlan) Addison-Wesley, London.
24. Riese, C. (1984). *Transformer fault detection and diagnosis using Rulemaster by Radian.* Technical Report, Radian, Austin, Texas.
25. Sammut, C. and Banerji, R. B. (1986). Learning concepts by asking questions. In *Machine learning* Vol. 2, (eds R. Michalski, J. Carbonell and T. Mitchell) Tioga, Palo Alto, California.
26. Schachter, R. D. (1986). David: influence diagram processing system for the Macintosh. In *Second Workshop on Uncertainty in Artificial Intelligence*, AAAI, Philadelphia.
27. Shapiro, A. (1987). *Structured induction in expert systems.* Addison-Wesley, London.
28. Stirling, D. A. and Buntine, W. L. (1988). Process routings in a steel mill: a challenging induction problem. In *AI'87, Australian Joint Artificial Intelligence Conference*, (ed. J. Gero) North-Holland.

10

Models of Inductive Syntactical Synthesis

J. Barzdin, A. Brazma, and E. Kinber
Computing Centre,
Latvian State University, Riga, USSR

1. INTRODUCTION

When designing or comprehending different algorithms we usually begin with considering a number of examples and then try to generalize from them. The aim of researches in inductive synthesis is to understand and formalize this process and eventually to design practical synthesizers.

Until the middle of the 1970's, the problems of inductive synthesis were studied mainly on the recursive-theoretic level: given the sequence

$$f(0), f(1), \ldots, f(m), \ldots$$

of the values of a recursive function f, it is necessary to restore an algorithm computing f (surveys [1, 2, 3]). Unfortunately, researches on this level did not produce many useful ideas for the construction of practical synthesizers.

In 1976 A. Biermann and R. Krishnaswamy [4] proposed a method of inductive synthesis from examples of full computation traces. This method does not put any significant restrictions on the class of synthesizeable programs leading to an exhaustive search. Furthermore, the presentation of full computation traces is inconvenient for the user.

For practical synthesizers the class of synthesizeable programs apparently should be limited. We have to look for inductive synthesis schemas (models) applicable to sensible problems within which the synthesis is effective and which are convenient for the user in the same time. Hence, the process of synthesis can be split into two steps—selection of the model appropriate for the given problem and the synthesis itself.

Here we will consider some of the inductive synthesis models developed in Riga. In our approach (inductive syntactical synthesis) the input information is regarded as a string of characters without any semantics. Synthesis in such an approach is based on the detection of purely syntactical analogies. The required program is synthesized in a form of a grammar describing all possible computation traces (sample computations), since to describe a program in fact means to give all its sample computations. Such a grammar can be considered as some non-traditional way of presenting a program (in fact, a program schema,

obtaining semantics only after interpretation of operators and predicates).

2. DOTS EXPRESSIONS (J. Barzdin [8, 9, 11])

Let us consider the algorithm computing the greatest common divisor (GCD) of two natural numbers X1 and X2. Let mod (X1, X2) be the remainder from whole division of X1 by X2. Then a possible way to describe the GCD algorithm might be the example shown in Figure 1.

```
Input   X1, X2
  Let   X3 := mod(X1, X2);
  Is    X3 = 0? Suppose it is not.
  Let   X4 := (X2, X3);
  Is    X4 = 0? Suppose it is not.
  Let   X5 := mod(X3, X4);
  Is    X5 = 0? Suppose it is.
Return  X4;
```

Figure 1. Sample computation of GCD.

Let us consider another example—a bubble-sort algorithm shown in Figure 2.

```
Input A:Array (1 . . . 4);
  If A(1) ⩽ A(2) then; else A(1) ↔ A(2);
  If A(2) ⩽ A(3) then; else A(2) ↔ A(3);
  If A(3) ⩽ A(4) then; else A(3) ↔ A(4);
  If A(1) ⩽ A(2) then; else A(1) ↔ A(2);
  If A(2) ⩽ A(3) then; else A(2) ↔ A(3);
  If A(1) ⩽ A(2) then; else A(1) ↔ A(2);
Return A;
```

Figure 2. Sample computation of bubble-sort algorithm.

From these descriptions (sample computations) evidently it is possible to restore the respective general algorithms. Therefore, a description of the algorithm by sample computations can be regarded as a program in some non-traditional programming language. How can the semantics of such a language be defined? A usual way to define the semantics of a language is by translation to another language with known semantics. Since here the intended language is that of examples, it is natural to define its semantics by means of inductive inference rules, restoring the general algorithms from their sample computations.

As a means for presenting general algorithms here we will use so-

called 'dots expressions'. The basic construction in the language of a dots expression is a dots term—any word of the following type:

$$\langle a_1 a_2 \ldots a_n \circ\circ\circ b_1 b_2 \ldots b_n \rangle,$$

where $a_1 a_2 \ldots a_n$ and $b_1 b_2 \ldots b_n$ are words in a given alphabet Σ and there exists such a $c \in \mathbb{N}$, that for all $i \in \{1, \ldots, n\}$ any of the two following conditions holds: either $a_i, b_i \in \mathbb{Z}$ and $|a_i - b_i| = c$, or $a_i = b_i$. For example

$$T_1 = \langle A1 \circ\circ\circ A5 \rangle,$$
$$T_2 = \langle A0(1) \circ\circ\circ A5(6) \rangle$$

are dots terms.

Substituting dots '$\circ\circ\circ$' in a natural way by the appropriate particular string and removing parentheses '$\langle$' and '$\rangle$' we obtain unfoldment $\mathrm{unf}(T)$ of the given dots term T. For example:

$$\mathrm{unf}(T_1) = A1A2A3A4A5,$$
$$\mathrm{unf}(T_2) = A0(1)A1(2)A2(3)A3(4)A4(5)A5(6).$$

Another basic notion is a dots string, which we obtain by concatenation of dots terms or embedding one term into another. For example

$$W_1 = \langle A1 \circ\circ\circ A5 \rangle \langle 1B \circ\circ\circ 6B \rangle C1$$
$$W_2 = \langle\langle A11 \circ\circ\circ A13 \rangle \circ\circ\circ \langle A31 \circ\circ\circ A33 \rangle\rangle$$

are dots strings.

For dots strings an unfoldment can be defined. To get an unfoldment $\mathrm{unf}(W)$ of the dots string W we gradually substitute all its dots terms (starting with the outermost) by their unfoldments. For example, to get $\mathrm{unf}(W_2)$, we first substitute the outer term to get a dots string

$$\langle A11 \circ\circ\circ A13 \rangle \langle A21 \circ\circ\circ A23 \rangle \langle A31 \circ\circ\circ A33 \rangle.$$

Then, after substituting the rest of the terms we obtain

$$\mathrm{unf}(W_2) = A11A12A13A21A22A23A31A32A33.$$

Dots expressions are (in a sense) dots strings depending on a variable (we denote it by K). Thus, for example

$$E_1 = \langle A1 \circ\circ\circ A\underline{K} \rangle \langle 1B \circ\circ\circ \underline{K+1} B \rangle C1,$$
$$E_2 = \langle\langle A11 \circ\circ\circ A1\underline{K} \rangle \circ\circ\circ \langle A\underline{K}1 \circ\circ\circ A\underline{K}\,\underline{K} \rangle\rangle$$

are dots expressions. We define the value $\mathrm{Val}(E, K_0)$ of the dots expression E for the given $K_0 \in \mathbb{N}$, as the unfoldment $\mathrm{unf}(E(K_0))$ of the dots string $E(K_0)$, obtained from the dots expression E by substituting all the substrings $\underline{K+c}$ ($c \in \mathbb{Z}$) by value $K_0 + c$. Thus, for example, $\mathrm{Val}(E_1, 4) = \mathrm{unf}(W_1)$ and $\mathrm{Val}(E_2, 3) = \mathrm{unf}(W_2)$. For an arbitrary $K_0 \in \mathbb{N}$, a value $\mathrm{Val}(W, K_0)$, is called a formal example of the expression E.

The essential element of the given model is the system of inductive

inference rules reconstructing general expressions by their formal examples. The system has to be complete in the following sense: if the formal example of the given expression is long enough, then the system of inference rules synthesizes an expression equivalent to the given one. The system of three rules: *folding-up*, *standardization*, and *generalization* described below is proved to be complete for the dots expressions (A. Brazma [7, 10]).

The rule of folding-up substitutes long 'regular' substrings of the type $a(1)a(2) \ldots a(K_0)$ $(K_0 \in \mathbb{N})$ by corresponding dots terms $\langle a(1) \circ\circ\circ a(K)\rangle$. The rule of standardization expands maximally and shifts to the right (in a sense) all dots terms of the given dots string. The rule of generalization is applied at the very end (i.e. when to apply any other rule is impossible) and it substitutes all 'large' numbers of the dots string by expressions of the type $\underline{K \pm c}(c \in \mathbb{N})$, where K is a variable. Thus, given the string

$$A1A2A3A4A51B2B3B4B5B6BC1,$$

the system synthesizes the following dots expression:

$$\langle A1 \circ\circ\circ A5\rangle\langle 1B \circ\circ\circ 6B\rangle C1$$

From the sample computation of GCD given in Figure 1, the system synthesizes a general algorithm of GSD (see Figure 3).

```
Input    X1, X2;
⟨Let     X3 := mod(X1, X2);
  Is     X3 = 0? Suppose it is not; ∘∘∘
  Let    XK := mod(XK − 2, XK − 1);
  Is     XK = 0? Suppose it is not;⟩
  Let    XK + 1 := mod(XK − 1, XK);
  Is     XK + 1 = 0? Suppose it is;
Return   XK
```

Figure 3. General algorithm of GSD.

Similarly, the system synthesizes the bubble-sort algorithm in Figure 4 by the sample computation of the bubble-sort algorithm given in Figure 2.

```
Input A:Array (1 ... K)
 ⟨⟨If A(1) ≤ A(2) then; else A(1) ↔ A(2) ∘∘∘
   If A(K − 1) ≤ A(K) then; else A(K − 1) ↔ A(K)⟩ ∘∘∘
  ⟨If A(1) ≤ A(2) then; else A(1) ↔ A(2) ∘∘∘
   If A(1) ≤ A(2) then; else A(1) ↔ A(2)⟩⟩;
Return A;
```

Figure 4. Bubble-sort algorithm.

3. WHILE-EXPRESSIONS CONTAINING INTERPRETED PREDICATES $(x_1 \leqslant y_1) \wedge (x_2 \leqslant y_2)$ (E. Kinber)

Loop conditions in the expressions (programs) of this language are interpreted predicates of the type $x \leqslant y$ and their conjunctions; these predicates do not occur in sample computations (in fact, they are reflected implicitly and are shown in bold type). Thus,

$(x:=0)$ **WHILE** $(x \leqslant y)$**DO**(bxx^+)

(x^+ denotes the assignment operator $x:=x+1$) is a program in this language, and

$b0b1b2b3b4b5b6b7$

is its value (formal sample computation) for $y=7$.

The standard sort-merge algorithm that merges ordered arrays $A[1{:}m]$ and $B[1{:}n]$ to an ordered array $C[1{:}m+n]$ can be written in this language as follows:

R:Input:A:ARRAY $(1 \ldots x_0)$, B:ARRAY $(1 \ldots y_0)$
C:ARRAY$(1 \ldots x_0 + y_0)$
$(x:=1, y:=1, z:=1)$
WHILE $((x \leqslant x_0) \wedge (y \leqslant y_0))$
DO(**CASE**: Is $A(x) \leqslant B(y)$? Suppose yes. Then
$C(z):=A(x), x^+z^+$
Is $A(x) \leqslant B(y)$? Suppose no. Then
$C(z):=B(y), y^+z^+$)
WHILE $(x \leqslant x_0)$**DO**$(C(z):=A(x), x^+z^+)$
WHILE $(y \leqslant y_0)$**DO**$(C(z):=A(y), y^+z^+)$
Output C.

The following sample computation naturally explaining the algorithm's behaviour is a formal sample of the program R.

Input: A: ARRAY $(1 \ldots 3)$, B: ARRAY $(1 \ldots 5)$,
C: ARRAY $(1 \ldots 8)$,
[Is $A(1) \leqslant B(1)$? Suppose yes.
Then $C(1):=A(1)$,
Is $A(2) \leqslant B(1)$? Suppose yes.
Then $C(2):=A(2)$,
Is $A(3) \leqslant B(2)$? Suppose no.
Then $C(3):=B(1)$,
Is $A(3) \leqslant B(2)$? Suppose yes.
Then $C(4):=A(3)$]
[$C(5):=B(2)$,
$C(6):=B(3)$,
$C(7):=B(4)$,

$C(8) := B(5),]$
Output C,

Special annotations (brackets [ ]) are used in this sample computation to specify approximate loop boundaries; their specification is apparently not difficult for a user.

A synthesis algorithm is developed which from a given system of annotated sample computations completely describing the programs behaviour, constructs the necessary WHILE-expression. It is proved that if the annotations specify loop boundaries correctly then the synthesis algorithm finds a program equivalent to the given system of sample computations.

The synthesis algorithm handles an arbitrary sample computation in polynomial time. However, the number of samples necessary for the synthesis can be exponential. Nevertheless, if the number of loops in a program is bounded, then the synthesis algorithm has a polynomial time complexity. It is interesting to generalize these results to the case when loop conditions are arbitrary Boolean expressions over predicates $x \leqslant y$.

4. FOR-EXPRESSIONS CONTAINING INTERPRETED FUNCTIONS (E. Kinber)

Programs in this language contain only FOR-loops. However, on the other hand, interpreted functions of the type $f(x, y_1, y_2, \ldots, y_n) \to x$ satisfying some monotonicity conditions are allowed. Using this model it is convenient, for example, to formalize the algorithm computing the sum of the first x natural numbers:

T: Input x; $(y := 0, z := 1)$
WHILE $(z \leqslant x)$**DO**$(y + z$, obtain $y; \mid z^+)$
Output y.

The word

Input 5;
[0 + 1, obtain 1;
1 + 2, obtain 3;
3 + 3, obtain 6;
6 + 4, obtain 10;
10 + 5, obtain 15;]
Output 15

is, in particular, a formal sample computation of T explaining the program's behaviour.

A synthesis algorithm is developed that, given a sufficiently 'representative' sample computation, synthesizes an arbitrary program in

this language. For instance, given the above sample computation, it synthesizes the program T. The algorithm works in polynomial (of the length of input samples) time.

5. GENERALIZED REGULAR EXPRESSIONS (g.r.e.-s)

(A. Brazma, E. Kinber [5, 6])

The language of g.r.e.-s is a general model which includes (in a sense) all models considered above (one can obtain these models by putting on g.r.e.-s different restrictions and fixing classes of interpreted predicates). G.r.e.-s are regular expressions over a set containing a set of variables $X=\{x, y, \ldots\}$, an alphabet $A(A \cap X=0)$, expressions $\underline{x+c}$, $x \in X$, $c \in N$, assignment operators $xa=y$, $x:=c$, $x,y \in X$, $c \in N$, and operators x^+ (adding of 1) and x^- (subtraction of 1). Examples of g.r.e.-s are

$$P_1 : (x:=0, y:=0)\,(ax^+x \cup by^+y)^*$$
$$P_2 : (x:=0)\,(ax^+x(y:=x)\,(by^+y)^*)^*.$$

To obtain a (formal) sample computation, say, for P_1, we compute an unfoldment of P_1, for example,

$$P_1 : (x:=0, y:=0)\ ax^+x\,ax^+x\,ax^+x\,by^+y\,by^+y \\ ax^+x\,by^+y\,by^+y\,by^+y$$

and then get the sample computation

$$a1a2a3b1b2a4b3b4b5$$

from P_1 making obvious calculations.

Statements of the type

$$\textbf{WHILE}\ (P)\textbf{DO}(C)$$

and

$$\text{CASE}(P_1 \to a_1, P_2 \to a_2, \ldots, P_k \to c_k),$$

in real programming languages correspond to the iteration (*) and union (∪), respectively.

For g.r.e.-s equivalence and inclusion problems are proved to be solvable. On the other hand, all recursive functions can be expressed (in a sense) by g.r.e.-s. A class of programs universal (in a sense) for all programs is defined, and a syntactical synthesis algorithm is developed for this class. The synthesis algorithm operates in a polynomial (of the length of input samples) time. The notion of the universality is defined in terms of closure with respect to a finite set of 'generalizing' transformations over programs.

An interactive synthesis algorithm is constructed for a wide class of g.r.e.-s: the algorithm from time to time can ask the user (formally the

oracle) whether various expressions are initial fragments of sample computations. The interactive algorithm can also synthesize universal programs.

6. THE GRAPHICAL EXPRESSIONS

The models of inductive synthesis considered above are linear in the sense that programs and examples are one-dimensional objects (words). The notion of a graphical expression is a generalization of dots expressions to graphs. For example, the graph given in Figure 5a is a graphical expression. Its value for $\alpha = 3$ and $\beta = 5$ is the graph given in Figure 5b.

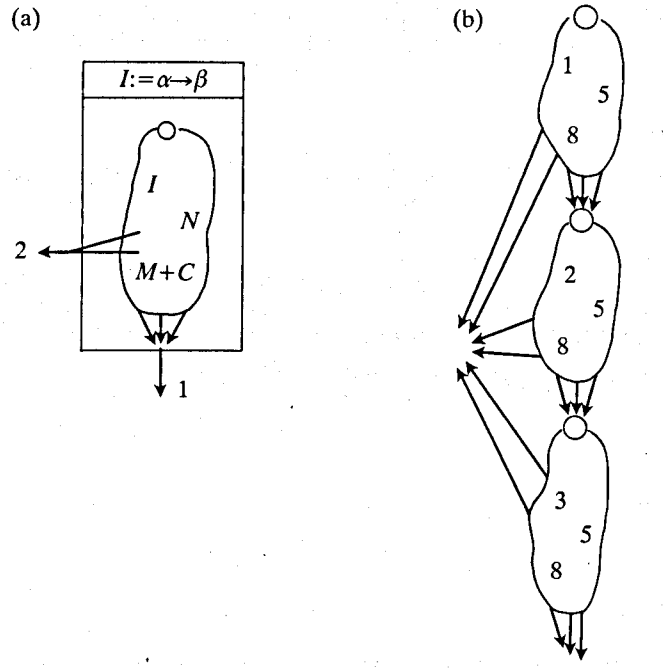


Figure 5. (a) Graphical expression. Its value for $\alpha = 3$ and $\beta = 5$ is the graph shown in 5(b).

For graphical expressions, a comparatively efficient heuristic method and experimental system of inductive synthesis has been developed by I. Etmane [12]. This system synthesizes a general algorithm (a graphical expression) solving a linear equation system of η equations for arbitrary η, from the sample computations solving a linear equation system of four equations.

The sample computations in the system of inductive synthesis considered are demonstrated directly on arrays monitored on the computer

screen by means of a light pen; Such sample computation is clear and visual: first we put on the screen input data, for example

$$A = (A(1) = 3, A(2) = 2, A(3) = 4, A(5) = 1),$$

then we put on symbols of necessary operations (for example >, ↔) and then create the string '$A(1) \leftrightarrow A(2)$' by touching with the light pen corresponding arrays $A(1), \leftrightarrow, A(2)$; at the same time the values of $A(1)$ and $A(2)$ on the screen are automatically interchanged.

This method allows us to present sample computations in many cases more easily and conveniently than respective general algorithms. (This approach was proposed first by A. Biermann [4])†.

A promising possibility for the application of inductive synthesis is program optimization. The basic idea here is to unfold the loops (i.e. substitute for them particular linear sample computations), then to optimize the linear programs obtained (their optimization is much easier than that of programs with loops). Finally we get back the loops from linear programs using inductive synthesis.

In [8] a different application of inductive synthesis—the synthesis of hypotheses about invariants of loops—is presented.

Finally, it should be noted that translation from all the models considered to a traditional programming language is a purely technical problem.

REFERENCES AND BIBLIOGRAPHY

1. Barzdin, J. (1974). Inductive inference of automata, functions, and programs. *Vancouver, an international congress of mathematicians*, **2**, pp. 455–60 (Russian).
2. Angluin, D. and Smith, C. H. (1983). Inductive inference: Theory and method. *Computing Surveys*, **15**, No. 3, pp. 237–59.
3. Klette, R. and Wiehagen, R. (1980). Research in the theory of inductive inference by GDR mathematicians—a survey. *Inform. Sci.*, **22**, pp. 149–69.
4. Biermann, A. W. and Krishnaswamy, R. (1976). Constructing programs from example computations. *IEE Trans. Soft. Eng. SE-2*, pp. 141–53.
5. Brazma, A. N. and Kinber, E. B. (1986). Generalized regular expressions—a language for synthesis of programs with branching in loops. *Theor. Comp. Sci.* **46**, pp. 175–95.
6. Brazma, A. and Kinber, E. (1985). A language for non-linear program synthesis containing while-loops. In *An intelligence formalization: semiotical aspects*, Kutaisi, pp. 173–6 (Russian).
7. Brazma, A. (1986). The decidability of equivalence for graphical expressions. In *Theory of algorithms and programs*, pp. 103–56. Riga. (In Russian.)
8. Barzdin, J. M. (1983). Some rules of inductive inference and their use for program synthesis. *Inf. Processing '83*, North-Holland, pp. 333–8.

†See also J. S. Collins (1964). The processing of lists and the recognition of patterns with application to some electrical engineering systems. Ph.D. thesis, University of London.

9. Barzdin, J. M. (1983). An approach to the problem of inductive inference. In *Applications of mathematical logics*, Tallinn, pp. 16–28 (Russian).
10. Brazma, A. and Etmane, I. (1986). Inductive synthesis of graphical expressions. In *Theory of algorithms and programs*, Riga, pp. 156–89 (Russian).
11. Barzdin, J. M. (1981). On inductive synthesis of programs. Lecture Notes in *Comp. Sci.*, **122**, pp. 234–54.
12. Etmane, I. (1985). An experimental system for the presentation of simple aids to the synthesis of programs by examples. Latvian State University—*Research Memorandum*, Latvia algorithm and program fund, No. IPO 017 (Russian).

OPTIMALITY AND ERROR IN LEARNING SYSTEMS

11

Deriving the Learning Bias from Rule Properties

J. G. Ganascia
Laboratoire de Recherche en Informatique,
Université Paris-Sud, France

Abstract

To guide learning, the 'learning bias' has to be defined with accuracy, even if it has to be modified when the results do not match expectations. This needs knowledge of the semantics of the various aspects of the learning bias: representation formalism, description language, and syntactico-semantical constraints applied to the learning assumptions. In this paper we try to show these various aspects and demonstrate how a new system CHARADE, provides some of these aspects with semantics, thanks to the introduction of the notion of 'system of rules'.

1. INTRODUCTION

Many authors have insisted on the importance of the description language and the generalization heuristics in induction. To illustrate this, let us use an elementary example (Figure 1).

Even in such a simple example, the number of generalizations possible for the scenes $S1$ and $S2$ is very high, for example,

(1) there are two objects on top of one another;
(2) the lower object is a polygon;
(3) there are two objects and one of them is a square;
(4) there is a striped polygon.

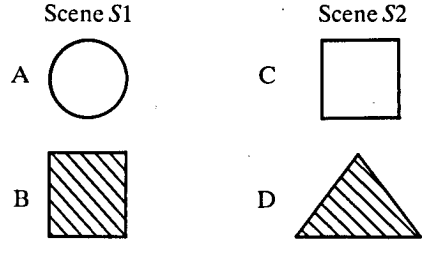


Figure 1.

Given this large number of potential intuitive generalizations, it is necessary to choose the most adequate, that is, that which in a given context will express best the common and relevant characteristics of the examples. A quick examination of our own behaviour shows that, on one hand, we are able to arrive at generalizations very quickly, even if they turn out to be incomplete, and that, on the other hand, we never consider all the generalizations possible. It seems as if, in a given situation, one generalization forces itself upon us. On reflection we may, of course, be able to question this first generalization and find a better one. Nevertheless, the number of generalizations considered never becomes very large. Besides, we may note that only the generalizations relevant to the goal are studied, and that in many cases, the difficulty lies more with the formulation of the generalization than with the recognition of commonalities. Lastly, we note that we are able to assess the accuracy of a generalization, whether we have formulated it or not. We thus have an intuitive notion of generality which allows us to assess the value of an intellectual piece of work, even if it is not our own, and we would personally be unable to achieve it.

In the context of machine intelligence, if the generality relation is independent of the subject who sets it forth, it must be possible to formalize this relation. But as ideas, or rather the propositions describing ideas, can be translated into a formal language, the relation of generality between ideas, or propositions, must also be translatable into formal relations between formulas of the description language. Such a formalization should cover all the possible relations of generality. Many papers (Michalski 1983, Mitchell 1982, Plotkin 1970, Kodratoff and Ganascia 1986, etc.) have shown the difficulties of elaborating a consistent theory of generalization. Here we shall mention briefly a few aspects of the generality relation, and their formal translation:

1. *Constant variabilization* if x is a variable, A a constant, P a predicate and $P(t_1, t_2, \ldots, t_n)$ an atom then $P(x, t_2, \ldots, t_n)$ is more general than $P(A, t_2, \ldots, t_n)$.
   We shall note it as $P(x, t_2, \ldots, t_n) \leqslant P(A, t_2, \ldots, t_n)$.
2. *Use of the dropping rule* if $P1$ and $P2$ are two propositions, $P1 \leqslant P1$ & $P2$.
3. *Climbing in a conceptual hierarchy* given an elementary conceptual hierarchy of explicit relations of generality between plane figures (Figure 4), POLYGON < SQUARE and thus the proposition (SHAPE = POLYGON) & (COLOUR = RED) is more general than the proposition (SHAPE = SQUARE) & (COLOUR = RED).
4. *Extension of boundaries* in the case of valued attributes on ordered sets: if SIZE is an attribute which has values over the set of integers, (SIZE $\geqslant$ 7) is more general than (SIZE $\geqslant$ 37) and thus, (SIZE $\in$ [−7,7]) $\leqslant$ (SIZE $\in$ [−5,5]).

Also, whatever the formal theory of generalization chosen, an order of priority as to the potential generalizations must be indicated. In fact, just as man does not explore all possibilities, and is able to limit his search to a minute part of the space of generalizations, even with the risk of omitting some, a machine cannot explore the whole domain either. Constraints are defined which, as they propagate in the space of generalizations, limit the research combinatorics. This set of constraints defines what T. Mitchell (1982) has named the 'learning bias'. Without a learning bias there is, strictly speaking, no real induction as the whole space of the potential generalizations has to be explored. On the other hand, if the learning bias imposes too many constraints, it is then possible that the best generalization is omitted. Moreover, the learning bias directly affects the nature and the quality of the generalization. Thus, it is frequently necessary to correct the learning bias through taking into account the results and the behaviour of the learning program (see Michalski 1983). An ideal learning program should itself be able to define and modify its own bias automatically. Utgoff's work (1986) moves in this direction when he studies the 'shift of bias for inductive concept learning'. However, a closer view shows that the notion of learning bias covers elements which widely differ; it includes the formalism of the expression language, its structure, and common sense syntax criteria (like the simplicity of a description). To be able to modify the learning bias quickly and correctly, whether manually or automatically, and to understand the exact scope of the modifications, the meaning of such modifications must be clear. Now, if the addition of descriptors always goes with some unintelligibility, conversely, the imposition of syntactical constraints upon the form of the generalization, such as the maximum number of terms in a conjunction (Michalski 1978), conforms only to very general intuitive notions without making clear the precise semantics of such constraints. The aim of this paper is to show that, provided we consider the learning process as the elaboration of a theory, that is, a system of rules, and not merely as the generalization of expressions, it is then possible to establish clear semantics for most classical syntactical constraints. At the same time, it is possible to define new constraints to meet the properties of the theory to be constructed. This is what has been done in the CHARADE system (see below).

2. LEARNING BIAS

2.1. Role of the learning bias

As we have seen, the notion of learning bias has been frequently mentioned in symbolic learning (Mitchell 1982, Utgoff 1986, Rendell 1986, Michalski 1983, etc.) to the point where it is possible to say that there is no induction without a bias. As a matter of fact, the formal definition of the generalization defines a domain of potential generalizations,

but, in symbolic learning, one cannot be satisfied with a mere formal limitation of the generalization space, as the calculation procedures of a good generalization must be constructed. Also, such procedures could not make a comprehensive exploration of the space of potential generalizations, because of its dimensions. Indeed, even if such an exploration could be envisaged, most of the generalizations obtained would be uninteresting, either because they would be unintelligible or because they could not be made operational. The learning bias is, thus, part and parcel of the data in a symbolic learning system, as, through imposing more or fewer constraints on the exploration procedure, it determines both the learning procedure behaviour and the nature of the induced generalizations. More schematically, any procedure of learning by detection of regularities can be represented as a search, constrained by the learning bias, in a space of potential generalizations.

2.2. Phenomenological approach to the learning bias

As far as the bias appears in the form of constraints on an exploration procedure, it displays similar characteristics. It can thus be, first of all, *more or less strong.* Too strong a bias might reject all satisfying assumptions: one that is too weak will let the search wander over too vast a space for full exploration to take place. There is thus a golden mean to be found. In addition, there are qualitative differences between biases that give qualitatively different results. For instance, in the example shown in Figure 1, according to whether the attributes define shape, colour, or geometrical relations and to the priorities assigned to them, so the results will differ in quality. As the bias determines the quality of learning, it is sound to consider how to modify a bias, after examining its results. Once these have been taken into account, the bias can be strengthened or weakened. Utgoff's work (1986) aims at automating a specific type of modification, weakening by extension of the description language. It might be advisable to consider other weakening techniques, and also nothing is said in this work as to the strengthening of a bias. This paper aims to establish semantics for certain aspects of the bias, with the hope of automating its strengthening.

2.3. Nature of the learning bias

An empirical approach to symbolic learning has led us to distinguish three different aspects of the learning bias: the representation formalism, the description language, and the syntacto-semantic restrictions. As we shall see here, specific constraints correspond to each of these features. The first are related to the notion of generality, the second to the actual limitation of a representation space, and the third to the exploration procedure of such a space. A strong correspondence can be noted between the learning model presented in the previous paragraph and the

breaking down of the learning bias according to its various aspects. The analysis can be refined to show that for each learning system input or output there is a corresponding learning bias. To demonstrate this, let us go back to the former model.

The formal definition of the notion of generality presupposes the existence of a formalism to represent knowledge which directly affects the nature of the generalizations that can be envisaged. Thus, in the case of propositional logics, generalization by constant variabilization makes no sense. The first aspect of the bias that we have isolated thus corresponds to the first input of a learning system, i.e. the generalization formal definition.

Once the formalism has been established, the elaboration of a learning set presupposes the existence of a description language admitting this formalism for its syntax. But the description language is structured; relations between descriptors must be made explicit as axioms, logical implications, or conceptual hierarchies. Moreover, hidden descriptors may appear in the final generalization, and lastly, it is possible that some descriptors present in the initial description may be deliberately excluded from the final generalization. This leads us (see Mitchell 1978) to define two description languages, a source language, where the examples are defined, and a target language, in which final generalizations are expressed. Although these two languages are quite different and the target language can be modified independently of the source language, there is a close link between the structure of these two languages which cannot be in contradiction. This does not change anything as to the nature of the second bias which can be reduced to the source language.

Lastly, the characterization of outputs must be translated into constraints on exploration procedure. Among such constraints, some only imply a modification of the target language, through restriction or extension of the source language, whereas others directly translate the syntactical properties of the assumptions to be formulated. This aspect of the bias is undoubtedly the easiest to comprehend. It is more susceptible to iterative modifications than any other as it does not touch the formulation of examples. Yet, in practice, it plays the major part as it makes the properties of the learning system's output explicit. It is then necessary that it should be clearly defined before each activation of the learning process. Otherwise, either the exploration procedure will work only on trivial cases, or it will implicitly limit the scope of the assumptions scanned, and no modification will be able to alter the arbitrary nature of such implicit rules.

We will now show how CHARADE (See Ganascia 1987a and 1987b) proposes semantics for certain syntax criteria of the bias; to do so, we must first introduce the principle on which CHARADE is based.

3. CHARADE: LEARNING WITH SYSTEMS OF RULES

3.1. CHARADE's functionalities

Designed to facilitate the man–machine transfer of expertise, CHARADE automatically builds up knowledge bases from the following:

(1) a *description language* covering a set of typed attributes and axioms expressing the domain semantics;
(2) a *set of examples* described in this description language;
(3) the description of the *expert system functionalities* for which the knowledge base is to be constructed.

CHARADE operates by empirical detection of regularities. Its main characteristics are that:

1. It takes into account the semantics of the domain expressed as axioms.
2. It can simultaneously generate certain rules and uncertain rules modulated by a plausibility coefficient.
3. It can translate the functionalities of the expert system to be obtained into generalization heuristics. The main original feature of CHARADE is that it provides the generalization heuristics with clear semantics.
4. It constructs 'systems of rules' which can be directly used by commercial inference engines, rather than simply isolated rules.

3.2. Construction of rules by generalization

To understand fully the incentives which have led to the implementation of the CHARADE system, it is useful to recall the characteristics of conventional systems of rule learning by generalization.

3.2.1. *Generalization from the specific to the general*

Michalski's INDUCE system (1983) is the most typical example of this type of approach and can be schematically expressed as follows:

Given
— A concept C
— A set of examples $E_1 \ldots E_n$ and of counter-examples $CE_1 \ldots CE_p$ of C
Stage 1: Construction of a generalization of $E_1 \ldots E_n$ discriminating $CE_1 \ldots CE_p$. Call it E_g
Stage 2: Construction of the rule $E_g \rightarrow C$

Although this technique has been confirmed by major successes, it can be criticized on several grounds. In fact, generally, the rules constructed

cannot be used directly by an inference engine, and whenever they are, it is not without a major impoverishment of the resulting expert system. Among the causes of such an impoverishment, we may quote:

(1) lack of rule chaining: $A \rightarrow B$, $B \rightarrow C$;

(2) the difficulty, or even impossibility, of introducing uncertainty;

(3) lack of consistency and economy for the knowledge base: the rules are considered as isolated from one another;

(4) the frequent use of heuristics during the generalization process which is not justified, as these heuristics have no semantics.

3.2.2. *Generalization from the general to the specific*

The building up of decision trees with the Quinlan method (1983) and its numerous successors (e.g. Quinlan 1986) is a good representation of this approach. To summarize, we start from:

— A set of classes exclusive from one another: $C_1 \ldots C_k$
— A set of examples $E_1 \ldots E_n$ for each class C_i
Stage 1: Construct a decision tree to classify all the examples E_i
Stage 2: Generate rules from the decision tree.

Just as we have criticized the strategy of constructing rules by generalizing from the specific to the general, we could also criticize the consequences of moving from the general to the specific as follows:

(1) no rule chaining: $A \rightarrow B$, $B \rightarrow C$;

(2) no accounting for the description language semantics;

(3) learning heuristics based on a numerical function;

(4) no accounting for the properties of the system of rules.

3.3. CHARADE: detection of regularities

To construct knowledge bases CHARADE empirically looks for regularities in the learning set (Ganascia 1987*a*). To do so, the set of parts of the learning set and the set of descriptor conjunctions are represented by two Boolean lattices. Besides the economy of representation that they offer, the properties of the lattices are used in the learning process to facilitate the detection of regularities. In fact, a regularity corresponds to a correlation empirically observed in the learning set. If all examples that have a descriptor d_1 in their description also have the descriptor d_2, it is possible to induce that d_1 implies d_2 in the learning set. The principle of induction used in symbolic learning consists of a generalization of this relation to the whole description space. Thus, these regularities must be detected. Two functions D and C are used. The first one, D, goes from

the descriptor's lattice to the example's lattice. To each description, it associates the set of examples of the learning set covered by this description. The other, C, relates to each subset of the learning set, that is, to each element of the example's lattice, the set of descriptors present in the description of all the examples. These two functions are represented in Figure 2.

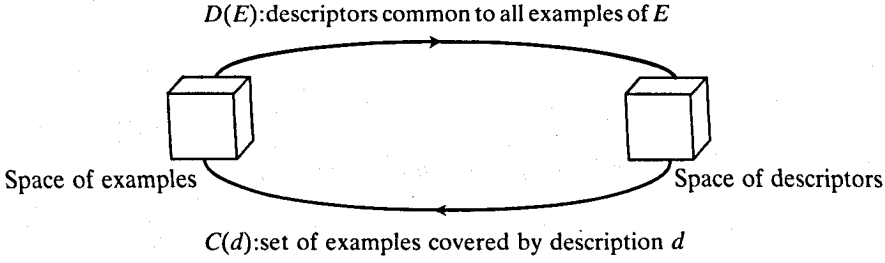

D(E):descriptors common to all examples of E
Space of examples
Space of descriptors
C(d):set of examples covered by description d

Figure 2.

Example Let us imagine a learning set of three examples E_1, E_2, and E_3:

$E_1 = d_1 \& d_2 \& d_3 \& d_4$
$E_2 = d_1 \& d_2 \& d_4 \& d_5$
$E_3 = d_1 \& d_2 \& d_3 \& d_4 \& d_6$

An empirical regularity can be detected between $d_1 \& d_2$ and d_4; in fact all the examples described by $d_1 \& d_2$ have also d_4 in their description. This is obtained with D and C as follows:

$DoC(d_1 \& d_2) = D(\{E1, E2, E3\}) = d_1 \& d_2 \& d_4$

This technique detects rules which are certain; however, when building a knowledge base, uncertainty must also be detected and translated; uncertainty may be due to either the poor quality of the learning set or rules which are uncertain by nature.

To translate such uncertainties, plausibility coefficients are frequently used. The latter are numbers in the range between $+1$ and -1 which alter the conclusions of the production rules, -1 representing the false and $+1$ the true.

It is easy to translate statistical regularities into production rules with plausibility coefficients. It is sufficient to calculate conditional probabilities with frequencies, and then to translate the result from the scale $[0,1]$ to the scale $[-1,+1]$. So, if D is a descriptor's conjunction, and if d' is a descriptor, the statistical correlation between D and d' is expressed by the following production rule: $D \rightarrow d'(\omega)$ where $\omega = 2*\Pr(d'|D) - 1$.

Example: to use once more the elementary example presented above, one obtains, among all the regularities detected, the following rules:

— Certain rule:
$d_1 \& d_2 \rightarrow d_4$ [$DoC(d_1 \& d_2) = D(\{E1, E2, E3\}) = d_1 \& d_2 \& d_4$]
— Uncertain rule:
$d_1 \& d_2 \rightarrow d_3$ ($\pi = 0.33$) [$\pi = 2^{*}\Pr(d_3 | d_1 \& d_2) - 1 = 2^{*}2/3 - 1$]

3.4. Exploration of the description space

Although we are able to detect and qualify regularities, we cannot detect all possible regularities; on one hand, the exploration procedure would be exponential (2^n conjunctions of descriptors, n representing the number of descriptors); on the other, the number of rules generated would be so large that they could not be managed by a conventional inference engine; last, such rules would be redundant. This would lead both to useless duplications and to errors in the case of rules with plausibility coefficients. There are thus two problems:

(1) how to limit the number of rules;

(2) how to limit the exploration.

Without describing in detail the mathematical formalism allowing the operations that we introduce, we give below the general lines on which the exploration procedure is based:

(1) use of the Boolean lattice structure of
— The set of parts of the learning set
— The description space (set of descriptors conjunctions)

(2) exploration from the GENERAL to the SPECIFIC;

(3) use of the properties of **rules**;

As far as the properties of rules represent the equivalence between rules, their implementation allows suppression of useless rules. This avoids redundancies at the same time that it limits the exploration of the description space.

Among such properties some are related to exact rules, reflecting the properties of logical implication. This is true for the two following properties:

If $\boldsymbol{a \rightarrow b}$ holds, then do not explore descriptions $\boldsymbol{a \& b \& \ldots}$, which implies that in our example, $d_1 \& d_3$, $d_2 \& d_3$, $d_1 \& d_2 \& d_3 \ldots$ are **useless** as $d_3 \rightarrow d_1 \& d_2$

If $\mathbf{C}(\boldsymbol{a \& b}) \subseteq \mathbf{C}(\boldsymbol{a \& c})$ holds, then descriptions of the type $\boldsymbol{a \& b \& c} \ldots$ are **useless**.

Other properties are related to approximate rules; they reflect the properties of plausibility coefficients:

If $\boldsymbol{a \rightarrow b(\omega)}$ holds, then rules of the type $\boldsymbol{a \& \ldots \rightarrow b(\omega')}$, are **useless** when $\omega \cong \omega'$.

(4) Use of the properties of the **system of rules** to be constructed.

Just as we introduced the properties of rules to avoid exploration of descriptions which, *a priori*, are not fruitful and to limit the creation of rules to those which are really useful, we use the properties of the system of rules that we want to build up. This is the most original aspect of CHARADE as, thanks to such properties, it establishes clear semantics for the generalization heuristics that are usually applied. At the same time, it introduces new heuristics. Thus, these properties become the operational characteristics of the expert system that we want to obtain. In fact, to build up a knowledge base, one must refer to the final state of the expert system required. Otherwise, the former would be just a collection of rules, many of them useless, some corresponding to random coincidences, others having no practical interest whatsoever. These characteristics are used, sometimes, to create knowledge acquisition tools to verify the relevance of the rules given by the user. An original feature of the CHARADE system is that it translates these characteristcs into constraints for the exploration procedure of the description space. An illustration is presented below.

1. *Goal* of the rules system (diagnosis, classification . . .). In the case of classification systems, if $C_1, \ldots, C_n$ are classes, whenever a rule of the type $a \rightarrow C_i$ is generated, descriptions more general than a (i.e. $a\&\ldots$) are **useless**.
2. *Minimum number of examples* covered by the rule premise: if $\text{card}(C(a)) \leqslant v$ then descriptions more general than a (i.e. $a\&\ldots$) are useless as $\textbf{card}(C(a\&\ldots)) \leqslant v$.
3. *A priori structure of the rule system* when an expert system is constructed, the structure of the system of rules that we want to generate is known *a priori*. Thus, for a therapeutic aid, whether in medicine or vegetable pathology, one knows beforehand that the rules go from the symptoms to the disease and then from the disease to the relevant therapy (see Figure 3). With this hierarchy, it is then possible to leave aside exploration of descriptions which correspond to a therapy and the study of regularities going from the diseases towards the symptoms or from symptoms to symptoms.

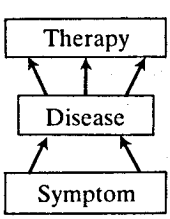


Figure 3. *A priori* structure of the rule system.

More generally, it is possible to introduce all the properties P such as when description D becomes useless because of the property P, then $D\&\ldots$ is useless because of the same property.

4. PRESENCE OF THE LEARNING BIAS IN CHARADE

We now consider what is frozen and what can be parameterized in CHARADE. Three essential aspects have been mentioned: the representation formalism, the source language, and the output properties which are translated both into the transformation of the source language and into the creation of syntactico-semantical constraints applied to the formulation of assumptions generated during the learning process. These are discussed below.

4.1. Representation formalism

In CHARADE, examples are represented as descriptors conjunctions. Each descriptor is a triplet (⟨Attribute⟩⟨Selector⟩⟨Value⟩).

The generalization can then be limited to the intersection of descriptors common to the various formulas, that is at the intersection of triplets (⟨Attribute⟩⟨Selector⟩⟨Value⟩) composing each formula. Working in propositional logics, there is no room for introducing the notion of constant variabilization in the generalization process, but one would still like to introduce the properties of descriptors or their mutual relations, which seems impossible through a mere use of elementary operations on sets.

For greater clarity, consider the two expressions $E1$ and $E2$:

$E1$ = (Size = 3) & (Shape = Square)
$E2$ = (Size = 5) & (Shape = Triangle).

Although no descriptor belongs simultaneously to both formulas, the examples they refer to have many aspects in common: they are polygons, they both contain red, and their size is between three and five. One would like to emphasize these common characteristics in the generalization process. We shall demonstrate in the following paragraph that, if we take into account the description language axioms, the intersection of descriptors is sufficient to obtain such characteristics.

4.2. The description language

As stated above, this can be considered as two consistent description sublanguages, the source language in which the examples are expressed, and the target language into which the generalizations are to be translated. The former example shows that, when reduced to the sole triplets (⟨Attribute⟩⟨Selector⟩⟨Value⟩) present in the examples, the target language is too limited to represent the generalization of $E1$ and $E2$.

This means that the source language must be expanded and all the implicit relations between descriptors must be made explicit.

Now, all these relations derive directly from the attribute's properties. It is then sufficient to express the properties of each attribute when defining the source language, to introduce them in the description of each example. But this requires, as a prerequisite, an accurate definition of such properties and of the modalities according to which they can be introduced into the example descriptions. To do so, we chose to classify the attributes according to their type, the type being related to the *domain of values* which can be associated with an attribute, to a *set of selectors* defining the modalities of such associations and to a set of *axioms* creating logical relations between the various descriptors built up with the help of an attribute.

In the example described above, two types of attributes can be identified: the ordered type to which the attribute *Size* belongs and the hierarchical type to which the attribute *Shape* belongs. Obviously, it is possible to define as many types as desired, such as monovalued, Boolean, etc., on the condition that they are properly described, that is the *domain of values*, the *set of selectors*, and a set of *axioms* is defined for each.

As an example, we give below the description of the two types mentioned above:

Ordered type:
Domain: set provided with a complete order relation
Selectors: $<$, $\leqslant$, $>$, $\geqslant$.
Accepted selector: $=$
Syntactical transformation: $(A=B) \rightarrow (A \geqslant B)\ \&\ (A \leqslant B)$
Axioms:
$[v1 < v2]\,(A \leqslant v1) \rightarrow (A \leqslant v2)$
$[\,]\,(A < v1) \rightarrow (A \leqslant v1)$
$[\,]\,(A1 < A2) \& (A2 \leqslant v2) \rightarrow (A1 \leqslant v2)$ etc . . .
Lower boundary: [optional]
Upper boundary: [optional]

Hierarchical type:
Domain: conceptual hierarchy
Selectors: $\leqslant$.
Accepted selector: $=$
Syntactical transformation: $(A=v) \rightarrow (A \leqslant v)$
Axioms: $[v2 = \text{father}(v1)]\ (A \leqslant v1) \rightarrow (A \leqslant v2)$ {addition}

Once they are well defined, the types allow us to characterize each attribute as a domain-associated type. Thus, *Size* is an *ordered type* for the integers or floating numbers accepting a lower and an upper bound.

Note that we establish in this case a distinction between the selectors $<$, $\leqslant$, $\geqslant$ and $>$ and the accepted selectors, $=$. The first ones are related to the properties of attributes via the axioms, whereas the second ones are considered only as a writing convenience. Each accepted selector is associated with a syntactical transformation which rewrites this selector containing descriptor into one or several descriptors containing the selectors of the corresponding attribute. For instance, the descriptor (*Size* = 3) will be rewritten as (*Size* $\leqslant$ 3) & (*Size* $\geqslant$ 3). It is only after such transformations have been made that it becomes possible to translate the properties of attributes with the axioms; in this case, the fact that the size of the two examples $E1$ and $E2$ is larger than three and smaller than five.

The *Shape* attribute is of a hierarchical type. The hierarchy of shape will be represented by a tree as in Figure 4.

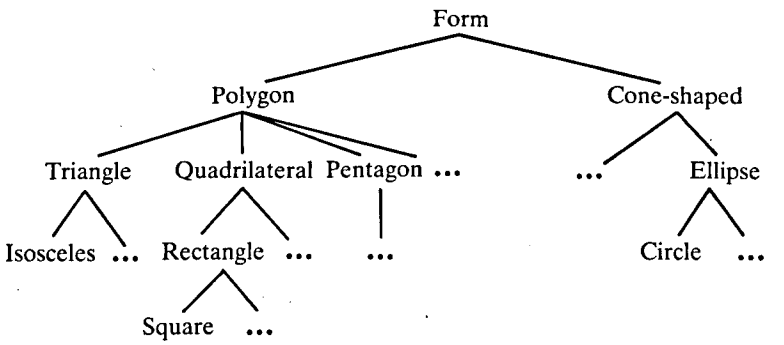


Figure 4. Hierarchy of shape represented by a tree.

In the hierarchical types, the axiom is labelled 'addition', which means that one adds to the example description all the conclusions that can be drawn from the axiom. In other words, one goes up into the hierarchy to introduce into the example description all properties inherited via the hierarchy. In the example $E2$, the descriptor (*Shape* = triangle) will be translated into (*Shape* $\leqslant$ triangle) which will generate the descriptor (*Shape* $\leqslant$ polygon).

To generate all the descriptors that are implicitly linked to one of the descriptors via an axiom, would be cumbersome, or even impossible in some instances (for example *Size*) where they are infinite.

Instead, we propose to generate only the descriptors likely to play a part in the generalization, that is, those that appear in other examples. Thus, if the two former examples, $E1$ and $E2$, are present at the same time, all the descriptors will be first transformed by the syntactical transformations, so that the only selectors to intervene in the descriptors will be those that are specific to the attribute to which they are related.

Simultaneously with this first transformation, the axioms labelled {addition} will be triggered.

After this first transformation $E1$ and $E2$ are described by:

$E1 = (Size \leqslant 3)\ \&\ (Size \geqslant 3)\ \&\ (Shape \leqslant \text{Square})\ \&\ (Shape \leqslant \text{Rectangle})\ \&\ (Shape \leqslant \text{Quadrilateral})\ (Shape \leqslant \text{Polygon})$

$E2 = (Size \leqslant 5)\ \&\ (Size \geqslant 5)\ \&\ (Shape \leqslant \text{Triangle})\ \&\ (Shape \leqslant \text{Polygon})$

Note that these descriptions are not comprehensive as the relations between descriptors do not appear. For instance the descriptor $(Size \leqslant 5)$: this descriptor ought to be present in both examples $E1$ and $E2$, as this is a piece of information that was implicit in the $E1$ description. Axioms must be used so that it becomes explicit. This is done during the second phase of transformation of the descriptions which computes all the relations that exist between the descriptors *present* on the lattice of examples. Once these relations have been calculated, the lattice of examples is completed with each of them. Thus, in the case of the descriptor *Size*, being of the ordered type, an axiom exists according to which: $[v1 < v2]\ (A \leqslant v1) \rightarrow (A \leqslant v2)$. Particularized for all the descriptors of the learning set, this axiom will generate the relation: $(Size \leqslant 3) \Rightarrow (Size \leqslant 5)$. As example $E1$ has $(Size \leqslant 3)$ in its description, it will be necessary to add $(Size \leqslant 5)$. After such transformation, the following descriptions of $E1$ and $E2$ are obtained:

$E1 = (Size \leqslant 3)\ \&\ (Size \geqslant 3)\ \&\ (Size \leqslant 5)\ \&\ (Shape \leqslant \text{Square})\ \&\ (Shape \leqslant \text{Rectangle})\ \&\ (Shape \leqslant \text{Quadrilateral})\ \&\ (Shape \leqslant \text{Polygon})$

$E2 = (Size \leqslant 5)\ \&\ (Size \geqslant 5)\ \&\ (Size \geqslant 3)\ \&\ (Shape \leqslant \text{Triangle})\ \&\ (Shape \leqslant \text{Polygon})$

For the latter, the descriptors common to $E1$ and $E2$ give a generalization $\text{gen}(E1, E2)$; in agreement with intuition one could have:

$$\text{gen}(E1, E2) = (Size \geqslant 3)\ \&\ (Size \leqslant 5)\ \&\ (Shape \leqslant \text{Polygon})$$

Without going into detail of the completion operations, it is possible to note that, if limiting oneself to the descriptors present in the learning set, there is neither a risk of explosion nor of making a loop. The axioms are used only to define the set of relations existing between the target language descriptors, without increasing it indefinitely.

Typing confers on the attributes precise semantics. But, and this is a negative consequence, these semantics are frozen. One would like to be able to choose a different point of view, or modify the generalization context or change the attributes' typing according to need. Although such a change of the attributes semantics is not automated, it is facilitated by the distinction between source language and target language: any

addition of axioms or any typing of attributes compatible with the source language is readily accepted by CHARADE and will be taken into account for the definition of the target language.

On the other hand, the first phase of completion of the example's descriptions can be taken advantage of to *add* new descriptors which, according to the expert's intuition, should play a part in the learning process. It is sufficient to give the name of such descriptors as well as their type and the procedures to obtain them. This is an important advantage. An example of its usefulness was given to us by an archaeologist: the elaboration of archaeology catalogues must meet two requirements: first to classify the objects newly registered, second, to create object typologies which must be at the same time concise, clear, and organized according to the major movements in history. The classification criteria vary according to authors and to eras; it is the task of the archaeologist to define new, more relevant ones. Unfortunately, assessment of the new criteria requires a considerable amount of work in reconstructing the typology of all existing objects. This is where symbolic learning systems like CHARADE are most efficient; it is only necessary to add the descriptors corresponding to the new criteria in the description of the registered objects, then to generate a new system of rules and to compare it with the old one. Thus, in a real problem, catalogues can take into consideration the shape, size, chemical composition, and ornaments in the classification of Bronze age axes. With CHARADE it is possible to test the relevance of other descriptors deriving, for instance, from set proportions. The latter have, up to now, been mentioned only as assumptions, their assessment being beyond our capability; but they might be linked to manufacturing constraints which would be interesting to examine.

4.3. Exploration of the description space

We have seen that CHARADE empirically detects regularities in an intelligent exploration of the description space. We have also demonstrated how the constraints on the exploration procedure derive from the properties of the rules and rules system that we wish to build up. On one hand, these properties represent a bias, in the meaning explained above, as they determine the nature of the results of learning at the same time that they constrain the induction procedure. On the other hand, they have a clear meaning as they describe the operational characteristics of the expert systems for which we intend to construct the knowledge base.

Thanks to the translation of the results properties as constraints on the exploration procedure, the number of descriptions to be studied is considerably reduced. The following example illustrates this.

Let D be the set of descriptors of the source language, with $\{d_1, d_2, \ldots, d_n\}$ n descriptors linked together by logical implications: $d_1 \Rightarrow d_2$, $d_2 \Rightarrow d_3$,

$\ldots, d_{n-1} \Rightarrow d_n$. Such links are frequent whenever there is an order relation between the values given to an attribute, for instance if 'height' is a numerical attribute, (height $\leqslant 2$) $\Rightarrow$ (height $\leqslant 3$) $\Rightarrow \ldots \Rightarrow$ (height $\leqslant 85$).

Let K be the maximum number of descriptions to be explored to scan, in the source language D, all the descriptions susceptible of giving rise to a regularity. One would like to calculate K', the maximum number of descriptors to be explored to scan the target language $D \cup \{d_1, d_2, \ldots, d_n\}$.

In the case where no semantical relation would link $d_1, d_2, \ldots d_n$, one would have: $K' = K*2^n$, as on the basis of each description of the original language, it would be necessary to derive 2^n descriptions. But, as, according to the constraints derived from the rules properties (Cf. above), if $a \Rightarrow b$, it is unnecessary to study descriptions of the type $a\&b$, none of the descriptions of the type $d_i\&d_j\& \ldots$ need to be studied as $\forall(i,j) \in [1,n] d_i \Rightarrow d_j \vee d_j \Rightarrow d_i$. Only the descriptions of the type A or $A\&d_i$ must be studied, A being a conjunction of descriptors of the source language D. As a consequence, the maximum size of the space to be explored is $K' = K*(n-1)$.

Thus, the structuring of the description space together with the provision of the properties of rules to be acquired minimizes the size of the description space which must be explored absolutely.

5. CONCLUSION

To conclude, the formalization introduced in CHARADE permits a better understanding of the learning bias semantics. This has three positive consequences.

First of all, the fact that the meaning of the learning bias is now made explicit precludes much wandering when adjusting the parameters that trigger the learning procedure. Then, the description of properties of the learning results characteristics as properties of rules systems provides the learnt systems of rules with an operational value. Finally, the study of the results of the learning procedure allows further refinement of the triggering parameter, and, if need be, the addition of descriptors, which, in the long run, should lead to the automation of the modification of the learning bias.

REFERENCES AND BIBLIOGRAPHY

Ganascia, J.-G. (1987*a*). *AGAPE et CHARADE: deux mécanismes d'apprentissage symbolique appliqués à la construction de bases de connaissance*, thèse d'état, Université Paris-Sud.

Ganascia, J.-G. (1987*b*). Learning with Hilbert cubes, in *Proc. EWSL*, Bled, Yugoslavia.

Hayes-Roth, F. and McDermott, J. (1978). An inference matching technique for inducing abstractions, *CACM*, pp. 401–11.

Kodratoff, Y. and Ganascia, J.-G. (1986). Improving the generalization step in learning in *Machine Learning: An Artificial Intelligence Approach, Volume II*, (eds R. S. Michalski, J. G. Carbonell, and T. M. Mitchell), Morgan Kaufmann, pp. 215–44.

Michalski, R. S. (1978). A planar geometrical model for representing multi-dimensional discrete spaces and multiple-valued logic functions. *Research Report UIUCDCS R-78-897.* Urbana, IL. Dept. Comp. Sci., University of Illinois.

Michalski, R. S. (1983). A theory and methodology of inductive learning in *Machine Learning: An Artificial Intelligence Approach* (eds R. S. Michalski, J. G. Carbonell, and T. M. Mitchell), Morgan Kaufmann, pp. 83–134.

Mitchell, T. (1978). *Version space: An approach to concept learning*, PhD Thesis, Stanford University.

Mitchell, T. (1982). Generalization as search, *Artificial Intelligence*, **18**, pp. 203–26.

Plotkin, G. (1970). A note on inductive generalization, in *Machine intelligence* 5, pp. 153–63 (eds. B. Meltzer and D. Michie), Edinburgh University Press.

Plotkin, G. (1971). A further note on inductive generalization, in *Machine intelligence* 6, pp. 101–24 (eds B. Meltzer and D. Michie), Edinburgh University Press.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games in *Machine Learning: An Artificial Intelligence Approach* (eds R. S. Michalski, J. G. Carbonell, and T. M. Mitchell), Morgan Kaufmann, pp. 463–82.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning*, **1**, pp. 81–106.

Rendell, L. A. (1986). A general framework for induction and a study of selective induction. *Machine Learning*, **1**, 177–226.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning, in *Machine Learning: An Artificial Intelligence Approach, Volume II* (eds R. S. Michalski, J. G. Carbonell, and T. M. Mitchell), Morgan Kaufmann, pp. 107–48.

Vere, S. A. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions, *Artificial Intelligence*, **14**, pp. 139–64.

Vere, S. A. (1981). *Constrained N-to-1 Generalizations*, unpublished draft.

12

Error Tolerant Learning Systems

C. Sammut†
The Turing Institute,
Glasgow, UK

Abstract

We consider the task of a robot learning in a reactive environment by performing experiments. A reactive environment is one where changes occur in response to actions. Actors other than the learner may be present in the world. The robot performs experiments by modifying the environment and observing the outcome. These observations lead to a collection of concepts which constitute a theory of the behaviour of the environment, also called a world model. An experiment may either increase confidence in a theory or refute a theory, but it can never prove a theory. Therefore it is possible that the robot will develop an inaccurate model of its world. This paper discusses a number of issues involved in finding and repairing faults in a world model. It also describes some preliminary results obtained from a learning program called CAP.

1. INTRODUCTION

Learning systems can be divided into two categories: single-concept learners and incremental learners. AQ (Michalski 1983) and ID3 (Quinlan 1983) are examples of single-concept learning systems. They produce one set of rules from one set of data and have no memory which permits them to add to a knowledge base by further learning. Incremental learning systems remember the concepts which they have learned and can use them for further learning and problem solving. Some examples are, CONFUCIUS (Cohen 1978) and Marvin (Sammut 1981). These programs build a model of their task environment through successive learning experiences which require interaction with the environment.

The task that we consider in this paper involves a program learning to control an agent in a reactive environment. This is an environment where changes occur in response to actions. Agents other than the learner may be present. As an agent accumulates experience, it constructs a world model or theory of behaviour which can be used to predict the outcome

†Present address: Department of Computer Science, University of New South Wales, Sydney, Australia.

of events and to determine what actions may be necessary to solve problems. In our discussion, a theory is a collection of concepts, where a concept is a description of a class of objects or events.

The concepts which an agent can learn are determined by the events experienced by that agent, so any concepts created on the basis of only a small amount of experience are likely to be less accurate than concepts developed on the basis of more extensive data. This is so because a small number of instances of a concept may not contain sufficient information properly to characterize the concept. Because of this, it may seem wise to postpone concept formation until sufficient data are available. This may not be possible for a number of reasons:

1. The agent may be required to perform some task even before it has acquired enough experience to build an accurate model of its world. For example, the system may be responsible for controlling the attitude of a satellite and learning to compensate for changes in atmospheric drag, changes in mass, and so on.
2. Unless the program builds a world model, it has no means of assessing the usefulness of the information it gains. Thus it is impossible to determine when 'sufficient information' for concept formation has been accumulated. The program is also unable to employ a directed search for further data since there are no criteria for evaluating the usefulness of information.

A learning system which postpones theory formation can acquire data either by passively observing the environment and other actors, or it may perform its own, unguided actions. With passive observation, there is no guarantee that the observations will be representative of the world unless an agent is acting as a teacher and showing the learner 'interesting' things. Thus it may be necessary to wait a very long time before sufficient data for theory formation have been accumulated.

Active learning requires the agent to perform experiments, that is to perform actions and note the changes in the world. A random choice of an action is as good as any when there is no theory to guide the learner. Unfortunately, random actions in a complex domain may produce instances of many concepts, giving the learner a confused mass of data. One of the benefits of constructing a partial theory while accumulating data is that the partial theory will suggest what kind of data to look for. If part of the theory can be identified as deficient then a strategy may be devised to seek the information necessary to repair the theory.

For incremental learning systems which operate in reactive environments, the biggest problem is maintaining consistency in a knowledge base which is constantly changing. Observation of the world can cause new concepts to be hypothesized or existing ones to be revised. Thus the world model, which is stored in the knowledge base, evolves over time.

In the remainder of this paper we discuss a number of aspects of knowledge base maintenance, including determining that an error in the world model exists, finding the error, and repairing it. We also discuss some work in this area including Hume's CAP program (Sammut and Hume 1987), and related work.

2. PROBLEMS IN INCREMENTAL LEARNING

Let us now try to be more specific about the nature of the errors that can be made by an incremental learning algorithm. First, because the system is incremental and therefore responsible for building its own background knowledge, we assume that all concepts to be learned must be describable in terms of concepts previously acquired. This assumption leads to two possible errors:

1. The system may observe an event for which there is no prior knowledge that will allow the system to make generalizations and thus learn.
2. The system may observe an event for which there is prior knowledge. However, the known concepts are too general to describe the concept of which this event is an instance.

Thus, an inadequate knowledge base can result in either too specific or too general a concept being learned. Interestingly, Shapiro (1981) notes that analogous errors can occur in pure logic programs (as well as non-termination). Later we will show how some of the techniques proposed for debugging PROLOG programs can be used to 'debug' the knowledge base of a learning system where concepts are represented by Horn clauses in first order logic.

An example of an error introduced into the knowledge base by over-generalization follows. Suppose the system is learning the preconditions for pouring liquids into containers. First it must learn what a suitable container is. This can be done by attempting to pour water onto a number of different objects. The following sequence of actions illustrates some of the pitfalls of learning by experimentation:

1. Water is poured over a closed box. This fails because the water ends up on the floor rather than in the box.
2. Water is poured over a cup and succeeds.
3. Water is poured over an open cylinder and this also succeeds.

What can explain these successes and failures? Any explanation which the learning system attempts must be in terms of what it already knows. If it knows about objects that have circular cross-sections, it may hypothesize that the precondition for pouring a liquid into an object is that the

object must have a circular cross-section, as do the cup and open cylinder. Of course, this theory will fail if the robot tries to pour water into a closed cylinder.

A more useful concept for learning about pouring is that of convex shapes. However, the robot may not have learned this concept before trying to learn about pouring. Therefore, it makes an over-generalization. Why should the robot attempt to learn something when it is not prepared for it? An observation of the world may have brought its attention to this task. For example, one mechanism that children use to explore the world is to imitate adults, adding some variations of their own. Observing what are assumed to be rational actors provides a good focus of attention for learning; unfortunately, the actors may be observed doing something more complicated than the child can understand. That is, it has not yet learned all the background concepts necessary to describe adequately what is seen. This is exactly the situation in the case of the robot learning about containers into which it can pour liquids.

2.1. Recognizing that an error exists

A world model is intended to predict the outcome of events in the world. A theory is clearly incorrect if an unexpected outcome occurs. When the robot pours water over a closed cylinder, it has a theory that predicts that the water will remain in the cylinder. When the water ends up on the floor, there is obviously something wrong. However, recognizing that an error exists and knowing what it is are not the same thing. For example, it is easy to see that a computer program has a bug, but locating the bug is much more difficult.

2.2. Locating errors in a theory

A theory can be thought of as a network of interconnected concepts. The shaded node, *E*, in Figure 1 may represent the concept of an object that can contain liquid. This is needed to establish the preconditions for retaining liquids in some container. This concept may in turn be necessary for knowing how to make a cup of tea, for example. What happens if, in the process of making a cup of tea, the tea is spilt on the table. Which part of the world model was responsible for the mishap?

One way of locating the problem is to trace through the execution of the plan that leads to the unexpected result, testing each concept that contributed to the plan. That is, we debug the plan. In logic programming languages, declarative descriptions can be executed as programs, thus the distinction between a concept and a plan is blurred. This is very helpful for our purpose since, by adopting a Horn clause representation of concepts, we can profit from experience gained in writing intelligent debuggers for PROLOG. Sussman (1973) developed a debugger for

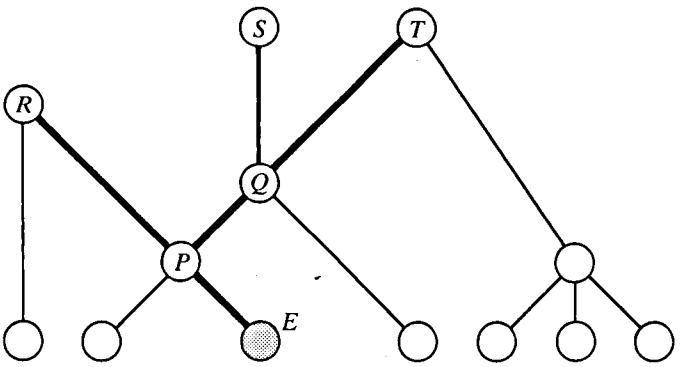


Figure 1. A theory is a network of concepts.

procedural progams. However, his program, called HACKER, used an *ad hoc* approach which required a library of possible fixes to common programming errors. We wish to minimize the number of assumptions necessary to locate errors in a theory. Horn clause logic gives a uniformity of representation which allows us to break errors into only three types.

Shapiro (1981) describes a program, MIS, for detecting logic errors in pure PROLOG programs. He claims that three categories are sufficient to characterize errors in pure logic programs:

(1) the program fails to terminate;

(2) the program returns an incorrect solution;

(3) the program fails to return any solution.

In the case of a faulty theory, return of an incorrect solution corresponds to an over-generalization in the theory since a concept has been used to describe an event that it should not be able to recognize. Failure to return an answer corresponds to a theory that is too specialized since it has failed to recognize an event it should have. Non-termination of a theory can occur when recursive concepts are involved. We will not dwell on the latter, but concentrate instead on theories that are too general or too specific. Note that some concepts in a theory may be too general while others are too specific.

To locate an error when an incorrect solution has been given (that is, the theory contains an over-generalization) Shapiro's debugging algorithm uses a method called *backtracing* to work backwards through the failed proof of a goal, searching for the procedure that caused the failure. In Figure 1, backtracing would begin with the last goal satisfied, that is, *T*. The debugger begins stepping back through the proof, i.e. down the dark path to node *Q*, then *P* if necessary, asking an oracle if the partial solution at each point is correct. If this is not true, then an erroneous

clause has been found. Note that the algorithm assumes the existence of an infallible oracle. In a reactive environment, the learning program can do without an oracle since the program is able to perform experiments to test a concept. Thus a failure suggests that the initial set of experiments which resulted in the formation of the concepts along the solution path was not extensive enough for at least one of the concepts. In the case of making a cup of tea, experimentation may identify the concept that describes preconditions for pouring liquids as faulty.

A concept that is too specific may prevent the program from being able to form a plan to achieve some goal. That is, the logic program that is supposed to satisfy the goal does not cover the initial conditions of the task. An attempt at debugging the theory can only be made when a correct solution has been seen, otherwise the learner has no indication that the task really is possible. A correct solution may be found, either by 'mutating' the current theory in the hope that the goal can be satisfied by the mutant, or by the learner observing another agent in the world performing the task. Shapiro's debugging method for programs that fail to produce an answer is equivalent to the second alternative, that is, the oracle supplies the correct solution. The debugger again tries to work backwards seeking clauses in the program that could have produced the given solution. Once such a clause is found, its body provides further goals that should be satisfied in order to arrive at the solution. The debugger considers each of these intermediate goals to see if they can also be produced by other clauses. Any goal that cannot be achieved indicates where the program or theory is deficient.

It should be noted that more intelligent search methods can be used to reduce the number of nodes tested. But while a method such as Shapiro's is very useful, it assumes that the learning system is able to suspend its current activities while it seeks the error in its theory. However, we noted earlier that there are applications where this is an unaffordable luxury. If the current theory works well in most cases and has failed relatively infrequently, it may be better to defer a thorough attempt at debugging in favour of persisting with the existing theory, while collecting information which will eventually point out the incorrect concept. For example, if over a period of time, all the concepts R, S, and T are noted to have errors then the system may conjecture that all the failures have a common cause and that there are only two concepts which could be that cause. Thus the search for the error in the theory has been reduced at the cost of tolerating more failures. A difficult problem is to decide whether it is more costly to stop and debug or to continue with some failures.

2.3. Correcting errors

After an erroneous concept has been detected what should be done to repair it? In his MIS program, Shapiro reasoned as follows: when a

program has failed to produce an answer, the program is too specific, so to repair it the debugger should generalize the program in order to cover the goals that cannot be satisfied. On the other hand, if a program produces an incorrect answer, i.e. it satisfies a goal it shouldn't, then it is too general and therefore the debugger should make the program more specific. This excludes the goals that were incorrectly satisfied. In order to perform these generalization and specialization operations, the program must have a *refinement operator*, which, given a term, will produce a set of terms that are minimally more specific than the original one. Thus, by recursive application, the refinement operator defines a language that is a subset of Horn clause logic. If MIS is to be able to create a correct program successfully, the refinement operator must be capable of generating the clauses necessary for the correct program. If the refinement operator defines too large a language, then the search time for the required clauses will be prohibitive. If the language is too small, then it may not be possible to generate the required clauses.

What do generalization and specialization mean when objects and events are represented by Horn clauses? A clause such as:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E$$

states that an object satisfying $A \wedge B \wedge C \wedge D \wedge E$ belongs to class X. When a clause describes an instance of a concept, all the literals in the clause are ground, that is, they contain no variables. A clause is generalized by replacing constants by variables and by replacing predicates such as A, B, C, etc, that describe some property of the object by other predicates that have more relaxed restrictions on the range of values which the property may assume. Specializing a clause introduces new restrictions on those ranges of values. MIS required that the predicates used in generalization and specialization were generated by a predefined refinement operator. To avoid having to know too much about the problem domain before starting, it is desirable to allow new terms to be introduced as required by the data. Suppose our learning system does not know a concept that will distinguish between a cup and a bowl on one hand and a closed cylinder on the other hand. However, by experimentation it is clear that there is some difference because the first two objects retain liquid poured over them while the closed cylinder does not. The learning system ought to be able to propose a new concept for objects that retain liquid. A method capable of this behaviour was first proposed by Sammut (1981). More recent related work by Muggleton is described below.

Muggleton's DUCE (1987) relies on a set of operators to *compact* the description of a set of examples to a simpler description. Each example is represented by a propositional Horn clause. Some operators preserve the equivalence of descriptions but reduce the number of symbols

required, while others produce generalizations. There are six operators in all and they are the basis for a method of anti-unification. Indeed, one of the goals of this work was to produce a complete inverse of resolution (see this volume). All six operators are necessary for the completeness of the theory, but pairs of operators are sufficient for induction. We will describe one such pair and refer the interested reader to Muggleton's paper (1987) for the complete description. A first-order version of DUCE, called Cigol (logiC backwards), has now been implemented (Muggleton and Buntine 1988).

Absorption. Given a set of clauses, the body of one of which is completely contained in the bodies of the others, such as:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E$$
$$Y \leftarrow A \wedge B \wedge C$$

we can hypothesize:

$$X \leftarrow Y \wedge D \wedge E$$
$$Y \leftarrow A \wedge B \wedge C$$

In fact, this is the generalization rule used by Sammut (1981) in his program, Marvin.

Intra-construction. This is the distributive law of Boolean equations. Intra-construction takes a group of rules all having the same head, such as:

$$X \leftarrow B \wedge C \wedge D \wedge E$$
$$X \leftarrow A \wedge B \wedge D \wedge F$$

and replaces them with:

$$X \leftarrow B \wedge D \wedge Z$$
$$Z \leftarrow C \wedge E$$
$$Z \leftarrow A \wedge F$$

Note that intra-construction automatically creates a new term in its attempt to simplify descriptions. At any time during induction there may be a number of applicable operators. The one chosen is the operator that will result in the greatest compaction.

As a robot performs experiments, its experiences may be stored as clauses representing observations. As this collection of clauses grows, it can be compacted, using the DUCE operators. This has the effect not only of reducing the storage cost of the information, but also of detecting patterns and introducing new terms into the knowledge base. For example, DUCE could easily detect the similarity of the cup and bowl when liquid is poured over them. The intra-construction operator would

introduce a new concept, that is, a new set of clauses whose heads have the same principal functor.

2.4. Maintaining consistency

Detecting and repairing an error in a single concept is one thing, but repairing an entire theory is another matter. Remember that in Figure 1, we envisaged a world model or domain theory as a network of interconnected concepts. Using a Horn clause representation, the head of a clause corresponds to a parent node and the goals in the body correspond to the children. These goals match other clause heads and form links to the rest of the network. Also in Figure 1, we imagined that one concept, represented by the shaded node, E, was in error. When the concept is repaired, what effect will that have on the concepts that referred to the old concept? Since P, Q, R, S, and T refer, directly or indirectly, to the erroneous node they must have been learned in the presence of the error. Are they, therefore, also in error or will correcting E alone correct them all?

When faced with the problem of ensuring the consistency of its knowledge base, two strategies are available to the learning system.

1. After correcting E, the system may test each of the concepts that depend on E. However, revising all of the concepts dependent on one that has just been modified could involve a lot of work if the network of concepts is very extensive.
2. The system may wait to see if any further errors show up. In this case, each concept will be debugged as necessary. Although more economical this method requires a method for tolerating errors if the program has been assigned a task that it must continue to perform.

It should also be noted that another source of errors in planning is noise. When a learning system is connected to a real robot, it cannot rely on the accuracy of measurements from vision systems, touch sensors, etc. Thus a plan may fail because the knowledge base does not accurately reflect the outside world. This being the case, the learning system must not revise its domain theory prematurely since there may not, in fact, be any errors. So the most prudent approach to error recovery is to delay revision of a domain theory until sufficient evidence has accumulated to suggest the appropriate changes.

Let us now give an outline of an error recovery strategy.

1. The robot learner is given a task which it is required to perform. However, its domain theory may be incomplete or incorrect.
2. In the course of performing its task, the robot's plan fails.
3. If the robot is unable to proceed by adopting another plan then it

must suspend working on its given task while it debugs its domain theory.

4. If an alternative plan is possible (for example, by reordering goals) then the new plan is attempted while storing the failed plan for future reference.
5. The robot cannot assume that the failed plan is incorrect since the cause of failure may have been due to noise. Therefore, as each plan is executed, a history of its performance is maintained; this includes the performance of the individual concepts that formed the plan.
6. The accumulation of histories is input for a DUCE style of learning system which effectively summarizes the performance of plans when it forms new concepts by generalization.
7. Since the learning program may generate alternative descriptions for the same concept, we must be able to resolve potential conflicts so that the next time a similar plan is to be created, the appropriate description is chosen.

In this final step, an assumption-based truth maintenance system (ATMS) becomes useful (de Kleer 1986*a*, *b*, and *c*). Alternative descriptions of the same concept represent different assumptions about the behaviour of the world. An ATMS provides a mechanism for carrying forward several lines of reasoning concurrently where each chain of inference is based on different assumptions. When a contradiction is encountered by one chain, it is knocked out and its assumptions invalidated. In our case, different lines of reasoning are replaced by alternative domain theories based on different concept descriptions. A failed experiment corresponds to a contradiction.

When an experiment does fail, we must not invalidate the concepts used in planning the experiment for, as mentioned earlier, the failure may be due to noise. Instead, we note the circumstances of the failure and augment the failed concept with a description of these circumstances. Several things could happen to the concept when this is done:

1. The description of the concept is modified to the extent that it becomes correct. If an alternative, correct description already existed, then the alternative domain theories of which these concepts were components, converge.
2. After several failures, there is no generalization which covers the circumstances of failure. In this case, the failures may be either attributed to noise or to some phenomenon not yet known to the system. In either case, nothing can be done.

An ATMS maintains the network of concepts that form a domain theory and stores dependencies which, when errors are found, will indicate

where other potential weaknesses in the theory lie. The ATMS also allows a learning program to experiment with alternative domain theories.

3. CAP

Hume's Concept Acquisition Program (Hume 1985, Sammut and Hume 1987) is a current example of a program working in a reactive environment. A reactive environment is one that responds to the actions of the learning program. This can be a real or simulated environment. In CAP's case we have a simulated world that contains two robots. One is under the control of the learning program which has little initial knowledge of the world. This is referred to as the *child* robot. The second robot already 'knows' about the world and can perform a variety of tasks. This is referred to as the *parent.* The child learns about the world by observing the parent performing some task and then using the observation to guide it in exploring its environment. Children often learn by trying to imitate the actions of adults. That is, when a situation arises which is similar to one where the parent has previously performed some action, the child may attempt the same action. Since the state of the world is unlikely to be identical to the initial state when the parent began its actions, imitation must also involve a degree of generalization.

To demonstrate how CAP works, we return to the example of learning the preconditions for pouring liquids from one container into another. Suppose the world consists of a solid cylinder and two cups, one with some liquid in it, the other empty. The parent robot's task is to pick up the full cup and pour the liquid contents into the empty one. At the completion of this task, the liquid is no longer in the original container, so it is not possible exactly to duplicate the same set of actions. If a child robot wishes to imitate the parent then it must be satisfied with a partial match of the starting conditions with some later state of the world. By partial match we mean that two states can be considered similar if some simple transformation can be applied to one state to turn it into the other.

Figure 2 shows the 'before' and 'after' states of the contents of cup A, being poured into cup B. Suppose the child wishes to imitate the action immediately after the parent has finished. A no longer contains the liquid; however, by comparing the descriptions of the original state and the final state we see that by substituting B for A we can use B as the source of the liquid. Similarly, substituting A for B allows A to be the destination.

Imitation based on a partial match is a useful way of learning. In this case, because A and B can be used interchangeably, the child will have learned the generalization that liquid can be poured into objects that are cups. Imitation can be viewed as an experiment whose purpose is to

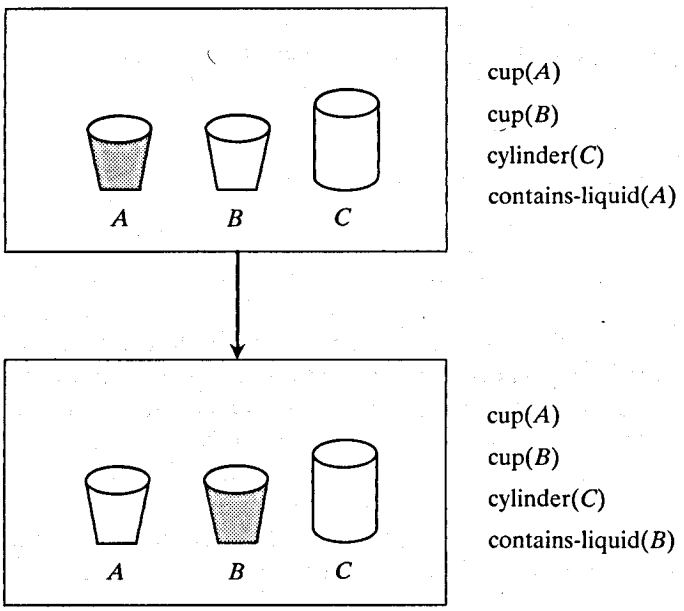


Figure 2. Finding a partial match between states.

confirm or deny a generalization. For example, after swapping A and B it can be predicted that the result will be that after pouring the liquid it remains in A. If the prediction is proved to be true then the generalization is confirmed. Let us now see how another experiment will fail to produce a predicted result but still yield useful information.

The partial match described above is obvious since one cup simply maps onto the other. However, there are more subtle similarities present in the scene. Assume that the concept:

$$\text{circular-cross-section}(X) \leftarrow \text{cup}(X).$$
$$\text{circular-cross-section}(X) \leftarrow \text{cylinder}(X).$$

is known to the system. That is, an object has a circular cross-section if it is a cup or a cylinder. This tells us that A, B, and C are all similar according to at least one criterion. Therefore, another possible substitution would allow C to be the destination of the pouring action. The previous experiment tested the effects of pouring liquid into another cup, thus permitting the generalization that any cup can contain liquid. Another experiment, this time with the cylinder, tests the generalization that objects other than cups can also contain liquids. Of course, this time the liquid does not stay in the cylinder. Thus the generalization is shown to be incorrect.

The child observes and records changes in the world as a sequence of states, where each state is represented by a description (in first order

logic) of the configuration of objects in the world. Suppose there is a sequence,

$$S_0; S_1; \ldots; S_N$$

and a current state, S. Although each S_i is a conjunction of atomic predicates, it is also useful to think of it as a set of predicates. Thus, a partial match can exist if there is some state, $S_m : 0 \leqslant m \leqslant N$ such that

$$S \cap \sigma S_m \neq \varnothing$$

That is, under some substitution σ states S and S_m share common terms in the state description. For example, if S_m consists of a full cup and an empty one and S consists of a full cup and a cylinder then a partial match exists with a substitution of the cylinder for the empty cup. However, in order for this substitution to work, it must have been recognized that cylinders and cups can be equated in some way. So before looking for a match, the system must first elaborate on the state description by using concepts, such as circular cross-section. This is done by treating the concept description as a set of forward chaining rules as described in Sammut and Banerji (1986) and is similar to the method used by Muggleton in DUCE.

The partial match permits CAP to propose the following task: since cups and cylinders are similar in at least one respect (they both have circular cross-sections) they may also be similar in their ability to contain liquids. Therefore, it should be possible to perform actions that will result in a liquid being poured into a cylinder, just as had been done with the cup (for which the cylinder has been substituted). This is the prediction, namely, that it should be possible to create a sequence of states in the world that corresponds to the sequence obtained through the matching process. The attempt to achieve the sequence in the modelled world is an experiment.

If an experiment has been concluded successfully, that is, the results match the prediction, then the child has grounds to propose a generalization. When the attempt to pour a liquid into another cup succeeds then it may be proposed that liquids can be poured into any cup. The pouring action, A_i, transforms a state S_i into another state S_{i+1}, written as

$$A_i : S_i \rightarrow S_{i+1}$$

Thus, S_i contains the preconditions for the action A_i. By generalizing S_i the applicability of the action is broadened. The next experiment, trying to pour the liquid into the cylinder, generalizes S_i even further. This generalization is incorrect; however, it can be recorded as an exception condition for action A_i. As other exceptions are encountered for A_i they may be combined with previously recorded cases. Thus, it is possible to

build up knowledge about when an action can be used and when it cannot.

Analysing why a particular substitution worked or did not work can lead to further experiments. If a particular generalization was successful, then it is worth looking for other objects covered by that generalization. If an attempt is made to broaden the generalization by substituting another object, and this fails, the difference between the object causing the failure and the previously successful objects helps to define the generalization more precisely. This refinement of the description can be achieved by trying to make the unsuccessful generalization more specific and performing another experiment.

An interesting side effect of this learning problem is that it provides a simple criterion for clustering objects into new, unnamed concepts. Objects form a cluster if they can be used in the same roles. Containers made of glass or plastic can both hold water, a brick and a table can both be used to support other objects. As an object is added to a cluster, a generalization may be performed in order to arrive at a concise description of the cluster. To perform this generalization we again refer back to DUCE's operators.

CAP's learning strategy can be summarized as follows:

1. The parent carries out a plan which results in a sequence of world states being created.
2. The sequence is compressed and stored.
3. CAP attempts a partial match between the current state and a stored state. The changes that result from the parent's actions are used as a focus for the search for a partial match.
4. Since a stored state belongs to a sequence, the nearest partial match is used to generalize the sequence starting from the matched state. The sequence thus generated attempts to predict the result of the experiment to follow.
5. A plan of action is inferred from the prediction sequence.
6. This plan is executed.
7. If the result of the experiment was as predicated then the preconditions for the actions in the plan are generalized, otherwise the exceptions conditions are generalized.
8. As long as there is nothing else to do (i.e. the parent is not doing anything that should be observed) the increasingly distant matches are used to create generalizations and experiments.

During experiments with CAP it was noted that the program could get itself into a state where further progress became impossible. If the water in all containers has been spilled then one of the basic ingredients of the

experiments is missing. To remedy this situation we 'cheat', but in an interesting way! Hume gave CAP the ability to keep more than one learning task running concurrently, so when progress is halted in one task, CAP can switch its attention to another. As well as the parent demonstrating how to pour liquid from one container to another, it also showed CAP how to create water by using a tap as a source of water. Once CAP has spilled all of its water and is unable to try pouring from one container into another, it switches its attention to learing about the use of taps. Of course, a side effect of this experimentation will be the creation of water in a container. So when the program has finished playing with taps, it can return to pouring water out of cups.

Sometimes, providing alternative tasks is not enough to prevent CAP from being blocked from further learning. Suppose, after experimenting with the tap, all containers have been filled, but the program still has not completed learning about pouring from one container to another. Even though its partial theory may tell CAP that any pouring action will result in spilled water, it may still do it if there is nothing more sensible that can be done. That is, if progress is impeded by a state that does not match anything in the program's knowledge base, CAP will perform a random and probably illegal action in order to bring about a useful new state of the world. In a sense we could say that the child throws a tantrum out of frustration, thus obeying the maxim: 'When in trouble or in doubt, run in circles, scream, and shout'!

4. RELATED WORK

Carbonell and Gil (1987) describe work on the PRODIGY general-purpose planner. Given an incomplete and possibly incorrect plan, they attempt to modify the plan when failures are encountered. An example of learning the correct plan for crafting the primary mirror of a reflecting telescope is given. (In fact, the example has the same characteristics as the classic blocks world planning problems.) In common with our earlier discussion, Carbonell and Gil point out that a plan which fails to meet expectations must contain errors. When a failure occurs, the system plans experiments to try to repair the domain theory. The theory should contain knowledge of the preconditions of each operator, the consequences, or post-conditions after applying each operator, and the objects to which it is appropriate to apply each operator. In this domain the operators include grinding, polishing, aluminizing, and cleaning objects.

The kinds of faults in the domain theory that the PRODIGY work has centred on are incomplete specification of pre- and post-conditions of operators and lack of knowledge about operator interaction. Since the domain theory is represented in first order logic, it is not surprising that

the method for planning experiments resembles Shapiro's program debugging in many ways. After a failure has forced the system to perform some debugging, the program examines the current state of the world and creates a new plan to achieve its original goal using the new domain theory. A potential hazard which the authors note is that replanning can cause a glass blank to be ground several times. If this is done too often, no glass will be left. This is similar to the problem that CAP encounters when it runs out of water during its pouring experiments.

5. CONCLUSION

We have discussed a number of problems related to incremental learning in a reactive environment. In particular, a learning system must be able to detect that it has an incorrect domain theory, and it must be able to locate the error and correct it while maintaining a consistent knowledge base. Some solutions to these problems have been suggested and an implementation of the CAP incremental learning program was described.

Acknowledgements

My thanks to Dave Hume, who designed and implemented CAP and to the members of the Turing Institute who have provided a stimulating environment in which to work during my leave from the University of New South Wales. Also to the Institute of Cybernetics of the Estonian Academy of Sciences for organizing a successful and fascinating meeting in Tallinn. My work at the Turing Institute has been supported by the Westinghouse Corporation.

REFERENCES

Carbonell, J. G. and Gil, Y. (1987). Learning by experimentation. In *Proceedings of the Fourth International Machine Learning Workshop* (ed. Pat Langley), pp. 256–66, Morgan Kaufmann, Los Altos.

Cohen, B. L. (1978). *A theory of structural concept formation and pattern recognition*, Ph.D. thesis, Department of Computer Science, University of New South Wales.

de Kleer, J. (1986*a*). An assumption based TMS, *Artificial Intelligence*, **28**, No. 1.

de Kleer, J. (1986*b*). Extending the ATMS, *Artificial Intelligence*, **28** No. 1.

de Kleer, J. (1986*c*). Problem solving with ATMS, *Artificial Intelligence*, **28** No. 1.

Hume, D. (1985). *Magrathea: A 3-D Robot World Simulation*, Honours thesis, Department of Computer Science, University of New South Wales, Sydney, Australia.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In *Machine learning: An artificial intelligence approach* (eds R. S. Michalski, J. Carbonell, and T. Mitchell), pp. 83–134. Palo Alto, Tioga.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Milan. See also this volume.

Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings 5th Machine Learning Conference*. Kafumann, pp 339–52.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application

to chess end games. In *Machine learning: An artificial intelligence approach* (eds R. S. Michalski, T. M. Mitchell, and J. G. Carbonell), Kaufmann, Los Altos, CA.

Sammut, C. A. (1981). Concept learning by experiment. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver.

Sammut, C. A. and Banerji, R. B. (1986). Learning concepts by asking questions. In *Machine learning: An artificial intelligence approach* (Vol. 2) (eds R. S. Michalski, T. M. Mitchell, and J. G. Carbonell), Kaufmann, Los Altos, CA.

Sammut, C. A. and Hume, D. V. (1987). Observation and generalisation in a simulated robot world. In *Proceedings of the Fourth International Machine Learning Workshop* (ed. Pat Langley), pp. 267–73, Kaufmann, Los Altos, CA.

Shapiro, E. Y. (1981). *Inductive inference of theories from facts*, Technical Report 192, Yale University.

Sussman, G. J. (1973). *A computational model of skill acquisition*, Ph.D. Thesis, MIT Artificial Intelligence Laboratory.

13

Use of Sequential Bayes with Class Probability Trees

D. Michie
The Turing Institute and the University of Strathclyde, Glasgow, UK

A. Al Attar
Attar Software Ltd, Leigh, UK

Abstract

For building classifiers from data, rule induction has established itself as an alternative to multivariate statistical approaches, including those of 'neural' computing. But modern rule-induction algorithms such as CART and C4 have not yet found a fully satisfactory way of discriminating logical from statistical forms of complexity in data. Systems of sequential Bayes rules offer a less *ad hoc* basis for combining probabilistic with rule-based approaches. In Evidencer, tree-structured rules are linked in a PROSPECTOR-like procedural hierarchy, and updated and processed according to a thresholding regime adapted from Wald's sequential analysis (1947). The resulting formalism steers a course between the oversimplifications of PROSPECTOR and the complexity of full multi-level Bayes, while retaining precise error bounds on decisions.

In comparative trials at the level of single-tree (unstructured) induction, Evidencer holds its own against state-of-the-art C4. Structured induction of multi-tree models has yet to be subjected to trial.

1. INTRODUCTION

Bayesian theory axiomatically defines probabilities as 'degrees of rational belief' (Good 1950), and hence offers attractions to the expert system designer who requires a consistent way of treating confidence and uncertainty.

The earliest use of Bayesian inference in knowledge engineering was in the expert system PROSPECTOR (Duda, Hart, and Nilsson 1976). PROSPECTOR 'learned on the job'—that is, run-time and hypothesis-update modes were closely interwoven. Pagallo and Haussler (1988) term such algorithms 'on-line'. In PROSPECTOR, goal-directed sampling from the task environment of attribute-values was used to adjust weights

(in the form of Bayesian odds-multipliers) of hypotheses embedded in a fixed multi-level inference structure. As a reference paradigm for such systems we have in mind not PROSPECTOR itself, but a later PASCAL-coded commercial shell, AL/X (Reiter 1980), in which separation of inference engine from domain-specific knowledge-base was clearer, and which expressed degrees of belief not as odds but as their logarithms—'plausibilities'. In plausibility-based systems, amounts of evidence show up as additive weights in place of the multiplicative 'factors' of odds-based systems (see Good 1950, 1983, and 1985).

In PROSPECTOR the simplified Bayes formula used for updating degrees of belief lacked the structure necessary to take account of conditional dependencies among evidence-sources, 'attributes' in rule-induction terminology. Inductive learning of the complete logic of decision could not in general occur, since inputs from attribute evaluation were combined arithmetically *via* adjustable weights (as also occurs in the fitting of multivariate regression equations), rather than *via* rule-based logical relations. As is also the case in neural nets, only weights could be learned. An analysis of neural net learning as a special case of multivariate statistical estimation has been made by Angus (1989).

Recent years have seen a variety of rule-based systems which derive and modify tree-structured rules from training sets of data, freezing the induced rule-trees for use as classifiers on further bodies of data; see Hunt *et al.* (1966), Friedman (1977), Quinlan (1979), Paterson and Niblett (1982, extension from logical to numerical attributes) and Breiman *et al.* (1984, the CART algorithm: first full treatment of class probability trees). In Pagallo and Haussler's terminology of machine learning, these are 'batch algorithms'. Integration of separately induced trees into a PROSPECTOR-like goal hierarchy was accomplished by Shapiro and Niblett (1982) and further developed by Shapiro (1987) into a generally applicable technique known as 'structured induction'. Here the outputs of lower-level decision-tree procedures are propagated up the hierarchy as inputs to higher-level procedures. But when these take the form of probabilities rather than of logical values, it has not been clear how they should be combined. Hence the scope of structured induction has remained restricted to domains for which every problem and subproblem is fully defined by a sufficient set of non-noisy attributes. With the object of removing this restriction, this paper considers the primary form of data-inducible tree-structured rule to be the so-called 'class probability tree' (Carter and Catlett 1987) in which a terminal node corresponds to a distribution of class-membership frequencies rather than to a definite membership assignment. We show how on a classical 'sequential Bayes' basis such trees can be derived from data in the form of **evidence trees** and hierarchically processed at run-time as **semi-decision trees**.

We shall in general preserve a terminological distinction appropriate to batch learning. We separate execute mode and edit (or 'learning') mode. According to mode, we speak sometimes of a procedural hierarchy (execute mode) and sometimes of an inference net (learning mode), meaning respectively dynamic and static aspects of the same thing. Likewise we speak either of decision procedures (execute mode) or of decision structures (learning mode); of subgoal invocation (execute mode) or of back-chaining links (learning mode). In Evidencer we do not generally admit of learning at run-time. In learning mode a data-set plays the role of a training set; in execute mode a data-set plays the role of a test set. Except for the special case described later of 'interrupt-revision' of a run-time classification, learning is off-line.

Thus use of training data to induce and update the structures that define procedure bodies constitute the system's learning mode. Execution of the procedural hierarchy on test data characterizes the run-time mode. With every procedure of the hierarchy is associated a hypothesis. The given procedure's run-time goal is to evaluate this hypothesis for the currently input data item and to output a belief-state. Arriving at a belief-state for the hierarchy's top-level hypothesis is identified with solving the top-level goal. A belief-state is obtained (see below) by comparing a computed plausibility with upper and lower plausibility bounds. The problem of propagating probabilities from lower to higher levels is thus circumvented in Evidencer. Outputs at every level are of type logical, from the three-element set {'accepted', 'rejected', and 'undecided'}.

PROSPECTOR supported uncertain inference, allowing certainty factors to be attached to the inputs and to the output of each procedure, and addressing the propagation problem by a proprietary method for calculating a certainty factor for the output of each procedure from the certainty factors of the members of its input set. The method was open to two objections:

1. Inputs that take the form of probabilities must in some way be discounted in order to take account of their degrees of uncertainty. These may arise, for example, from error in instrument readings, or in approximations used in computing or in estimating their values, including their inherently probabilistic status when such inputs are themselves the outputs of other Bayesian procedures in the hierarchy. To do this PROSPECTOR used an *ad hoc* interpolation procedure which lacks formal justification.

2. Even without uncertainty of inputs, the combining method used, sometimes called 'naïve Bayes', is only correct when all the data item's attribute-values that are input to the procedure are mutually independent with regard to their effects on the procedure's output

degree of belief. For general validity, 'conditional Bayes' must be substituted.

The above two difficulties subsequently received corrective treatment at the hands of various authors, culminating in Pearl's (1988) definitive account of Bayesian inference nets. Evidencer resolves them in a different way, namely by imposing as a constraint the knowledge engineering principle of inferential transparency. This demands that the inference net should be representative of the kind of conceptual complex that a domain specialist might feasibly execute 'in the head' or might communicate to a fellow-specialist.

In Evidencer arithmetic is accordingly minimized, and at run-time virtually eliminated. Within a back-chained inference net each node presents a separate induction problem, and from a training set of pre-classified data-items, an evidence tree is generated. From each such tree a thresholding process extracts a classification tree. At run-time the latter is executed on new data so as to map each item to one of the classes 'accepted', 'rejected', 'undecided'. We set out below the Evidencer algorithms, first for inducing the evidence tree, and then for extracting the classification tree. We also refer to the latter as a 'semi-decision tree': only two of its three output classes are strictly decisions.

2. INDUCTION OF EVIDENCE TREES

In addition to the above treatment of inter-procedure parameter passing, Evidencer makes a notional distinction between a **supplier** who manufactures a generic evidence tree from whatever data he or she regards as suitably representative, and a **user**. The latter derives from the supplied product a run-time structure (the semi-decision tree) customized to conform to his or her own data and to his or her own estimates of the relative costs of different kinds of run-time classification error. Otherwise the only substantive difference from the induction methods of Quinlan and of Breiman *et al.* lies in the substitution of the maximization of expected utility for entropy-minimization. Taking weight of evidence as a 'default' utility (see Good 1979, 1989), we replace the Quinlan–Breiman information measure by the *expected weight of evidence* as the basis of tree growth. This quantity in Bayesian probability theory is defined as follows.

Let $P(H)$, $P(H)/(1-P(H))$, and $\log P(H)/(1-P(H))$ be respectively the prior probability, the prior odds, and the prior plausibility of a hypothesis H (strictly written $H(x)$; the argument corresponds to the data item currently input for classification). Then the posterior plausibility of H in the light of a sampled event E is:

the prior plausibility of H plus the weight of evidence in favour of H yielded by the observed value of E,—or in symbols:

$$\text{plaus}(H|E) = \text{plaus}(H) + w_H(E)$$

$$\text{where } w_H(E) = \begin{cases} \log P(E^+|H)/P(E^+|\textbf{not } H) \text{ if } E = E^+ \\ \log P(E^-|H)/P(E^-|\textbf{not } H) \text{ if } E = E^- \end{cases} \quad (1)$$

Denoting the two alternative values of $w_H(E)$ by w^+, w^-, the *expected* weight of evidence from observing E is

$$P(E = E^+) \times w^+ + P(E = E^-) \times w^- \quad (2)$$

The 'sequential Bayes' extension considers the act of observing E's value as the most recent in a sequence of such events, giving:

$$\text{plaus}(H|E_i, (E_1, \ldots E_{i-1})) = \text{plaus}(H|E_1, \ldots E_{i-1}) + w_H(E_i|(E_1, \ldots E_{i-1})),$$

or in words:

> given a sequence of observations $E_1, E_2, \ldots E_i$, H's plausibility after E_i is equal to H's plausibility after E_{i-1} incremented by the weight of evidence contributed by E_i conditional on the pre-occurrence of $E_1, E_2, \ldots E_{i-1}$.

It is worth noting that H's plausibility after E_i must also equal H's *plausibility prior to* E_1 incremented by the total weight of evidence contributed by the *joint occurrence of* $E_1, E_2, \ldots E_i$. This equivalence is easily verified from the axioms of probability. It should also be noted that weight of evidence is equal to plausibility gain, and we shall use the terms 'evidence tree' and 'plausibility-gain tree' as interchangeable.

The above leads naturally to the following recursive algorithm (Michie 1989), which constructs an evidence tree from a training set C of pre-classified attribute-value vectors. We relax the restriction to two-valued attributes and allow multiple alternative outcomes, i.e. $E^1, E^2, \ldots E^m$, instead of just E^+ and E^-.

Given a collection C of vectors, each pre-classified as H = true or H = false:

1. If C is empty then label C EMPTY; otherwise
2. if C is not partitionworthy, then label C with its plausibility, estimated as

   log (size $\{x|x$ in C **and** $H(x)\} + 1) - \log$ (size $\{x|x$ in C **and not** $H(x)\} + 1)$;

3. if C is partitionworthy, then

use a selectable attribute E to partition C into subsets,
those with value E^1 into C^1,
those with value E^2 into C^2, etc.;
form a tree rooted in C and apply the same process to each of the sub-collections $C^1, \ldots C^m$.

The plausibility estimate of step 2 employs a smoothing device, 'Laplace's Law of Succession', to correct bias inherent in face-value estimation from small-sample frequencies.

Next we define 'partitionworthy'.

C is partitionworthy iff C is mixed and a significant candidate E exists.

C is mixed iff at least one member is pre-classified as true and at least one is false and the members of C do not have identical attribute values.

E is a candidate unless it already occurs in the path from C to the root, or unless its application to C would generate a subset of size less than m (a user-supplied pruning parameter).

E is selectable iff it has the highest expected weight of evidence among significant candidates.

Details of estimating prior probabilities and expected weights of evidence from the training set, and also of what distinguishes 'significant' members of a set of candidate Es, are in Michie (1989). Note that if, in contrast to the less developed approach of that paper, we wish to sharpen the distinction between evidence tree and semi-decision tree, then the above algorithm can be simplified by declaring 'significant' to be vacuous. At the expense of possibly growing an incoveniently ramifying tree, **all** candidates Es are then treated as significant. The effect is to concentrate pruning into the next phase (the 'setup operation'), in which the evidence tree induced by the supplier is customized to the user's task environment and economic criteria.

3. THE SET-UP OPERATION

Figure 1 illustrates the form of plausibility-gain, or 'evidence' tree obtained by zeroing the root of the above process on the plausibility scale. Its extraction from the training data, and conversion from plausibility scale to plausibility-gain scale, marks the close of what may be termed the 'laboratory phase'. The operational environment to which it is to be transplanted may or may not reproduce the relative frequencies of H and **not** H on which laboratory estimation of the prior log-odds was based. Yet the evidential structure implicit in the tree may still be

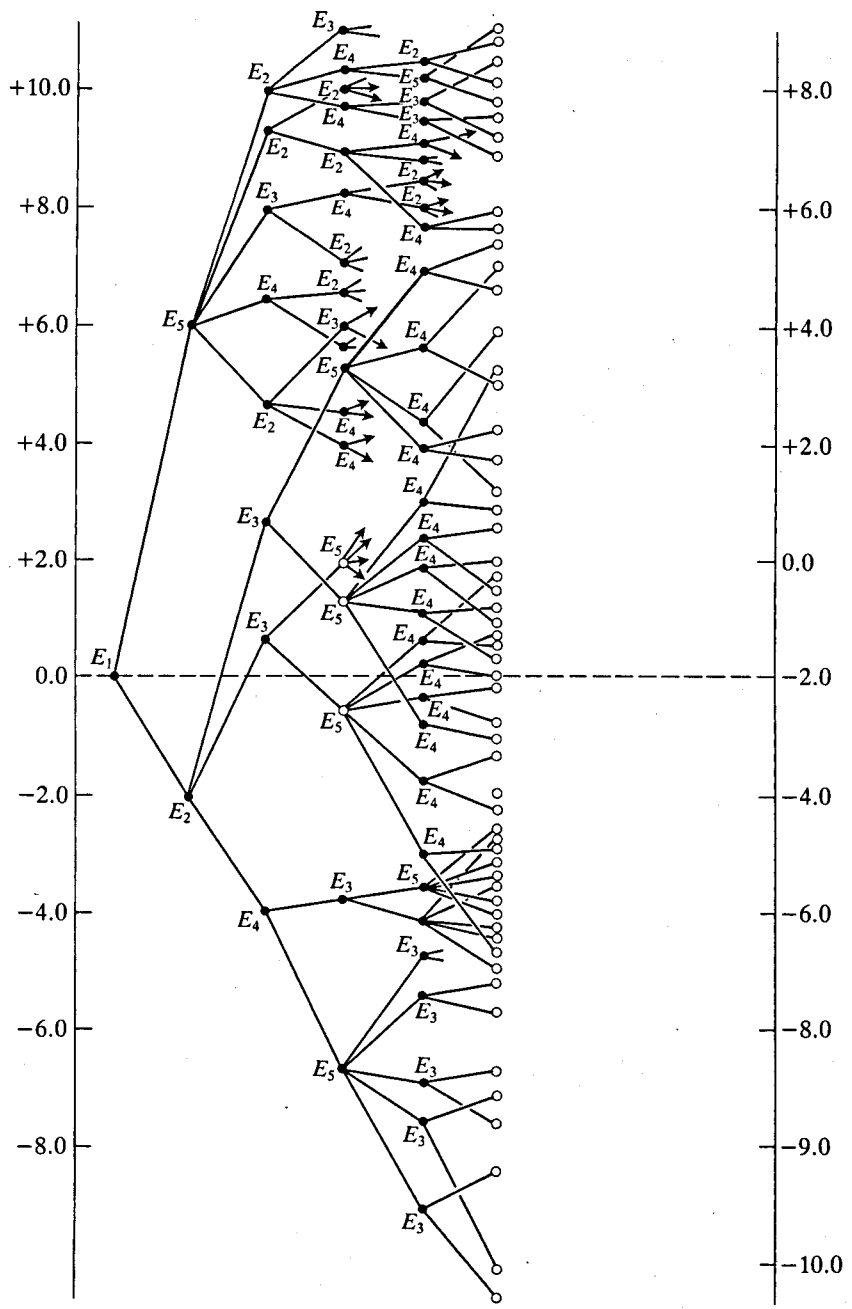


Figure 1. When read off against the left-hand scale, the above is an evidence tree (plausibility-gain tree). After conversion to the right-hand scale it becomes a plausibility tree (see text).

dependable. As can be seen from inspection of the Figure, the user is supplied with a normalized version of the tree, rooted at the zero point of the log-odds scale, an 'evidence tree'. To convert it into the semi-decision form showed in Figure 2 the user does as follows:

Step 1 He or she re-roots the tree to the point on the plausibility scale indicated by his or her own frequency data. This resembles the re-calibration of an instrument on transfer from bench to field. The result is a plausibility tree. In the example, prior odds of 1/100 are assumed.

Step 2 The user sets decision thresholds θ_1, θ_2 on the basis of a locally constructed loss function. Such a function expresses on a utility scale the expected loss when H is accepted but is false, and the expected loss when H is rejected but is true. He or she may additionally allow for the expected loss associated with an 'undecided' outcome (the cost of referral). θ_1 and θ_2 are chosen so as to optimize the user's expected cost-benefit.

Step 3 Branches intersected by θ_1 and θ_2 have their dependent nodes defined as terminal (i.e. 'leaves') with attached labels ACCEPTED and

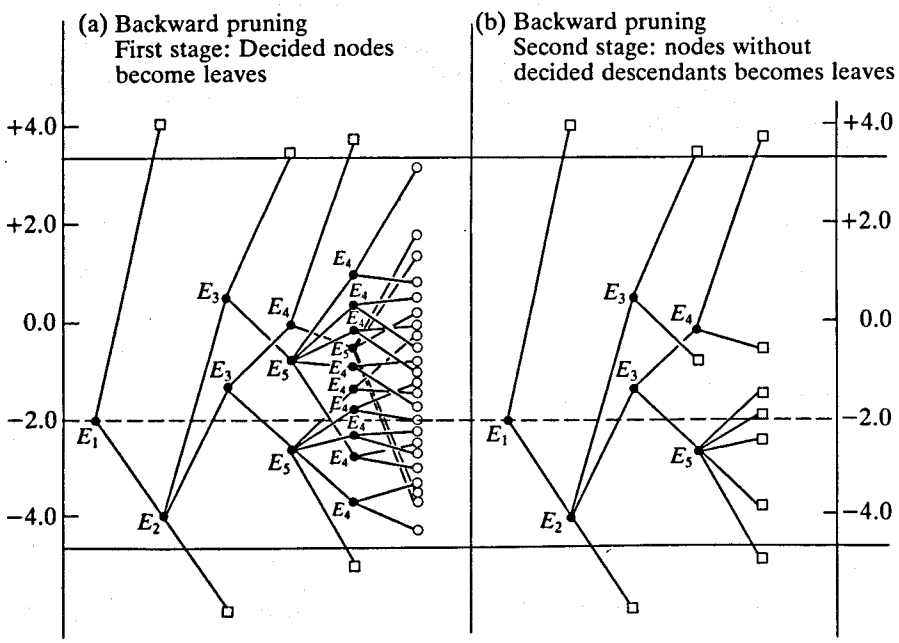


Figure 2. Successive results of backward pruning of a plausibility tree. The latter is obtained (see Figure 1) by rooting the supplied evidence tree at a point on the plausibility scale (right-hand of Figure 1) corresponding to the user's own frequency data. (a) Semi-decision tree from threshold-pruning only of the plausibility tree. (b) Semi-decision tree with three positive leaves, two negative leaves, and six undecided leaves. $\theta_1 = +2.67$, $\theta_2 = -4.33$.

REJECTED, all other leaves being labelled UNDECIDED. A further step of backward pruning is then applied to the latter. Wherever a node is found, all of whose terminal descendants are UNDECIDED, these descendants are deleted and the UNDECIDED label moved back to the ancestral node.

Step 4 The resultant semi-decision tree is compiled into executable code to form a run-time classifier. The tree is now ready to be run on new data sampled from the same source as that used for the re-calibration step.

Functionalities of the products of successive stages are as follows:

induction procedure: training sets → evidence trees

set-up procedure: evidence trees × prior plausibilities × loss functions → classifiers

run-time procedure: classifiers × test sets → classified sets

4. RUN-TIME

Although at run-time internal track is kept of plausibilities, semi-decision trees are classifiers and return only belief states. With every hypothesis is additionally associated a truth-value, 'true', 'false' or 'unknown'. A case, with belief-state set to ACCEPTED, UNDECIDED or REJECTED and truth value 'unknown', may subsequently acquire a truth value from a run-time oracle. For example, after forming the positive belief-state, or 'opinion', that a loan applicant is creditworthy, the system may later acquire information that the applicant has defaulted.

Conflict is resolved by the compatibility table of Table 1. 'Interrupt revision' of this type is the only learning that occurs at run time, and is effected by incrementation of exception lists attached to the leaves of semi-decision trees. If during routine use it becomes desirable to generalize over accumulated exceptions so as to revise the tree structure, re-entry to learning mode may be made. An analogy for this model of expert

Table 1. Compatibilities between belief-states and truth-values encountered at run time.

	'true'	'unknown'	'false'
'accepted'	**yes**	**yes**	**no**
'undecided'	**no**	**yes**	**no**
'rejected'	**no**	**yes**	**yes**

yes means compatible; **no** means incompatible. On re-entry to learning mode the system can restore compatibility by revising belief-states.

decision may be drawn with the hierarchical structure of subcommittees in a judicial process. Each subcommittee is called on by its immediately superior committee to furnish a considered **opinion** (belief-state) on some aspect of the case. Opinions at every level, once issued, are allowed to stand, and the top-level opinion, or 'verdict', is implemented. So although the internal process of each committee is known to involve uncertainty, its output—H accepted, H rejected, H undecided—is treated as firm, i.e. as a 'finding'. Subsequently truths at variance with the top-level verdict, or with a subcommittee finding on which it was predicated, may become known to the implementing authority and result in *ad hoc* rectifications (for example, judicial pardons in criminal law). Accumulation of many such cases may, moreover, lead to procedural revision. This can be achieved if the errant committee 're-enters learning mode' and reviews the rectified file.

Associating an exception list with each computed decision is similar to the 'rote' of a Pop-2 memo function (Michie 1968, van Emden 1972). A revised value entered into the list supersedes the value that would be computed by the rule on an identical case. A similar facility is found in the Mathematica system (Wolfram 1988).

In summary, PROSPECTOR-like propagation of probabilities is replaced by the more transparent process of passing discrete-valued belief-states, or 'opinions'. Procedural nets whose nodes are Bayesian semi-decision trees support run-time calculations on a logical rather than arithmetical basis, yet maintain an approximation to probabilistic reasoning. As a simulation of expert inference this offers a way of doing structured induction in probabilistic, noisy, and underspecified domains. Error bounds are maintained in the form of the θ_1/θ_2 threshold.

How well does it work in practice? A complete answer requires tests of the following:

(1) at the level of the individual tree: use of the expected weight of evidence in place of entropy and of Wald-type thresholding as a basis for pruning;

(2) at the inter-procedural level: propagation of outputs of uncertain inference in the form of 'opinions';

(3) (perhaps most controversial) the presumption that logical and evidential dependencies in a population are less labile under demographic shifts and other mutabilities of sampling than the frequencies themselves. It is this presumption which underlies the user-customization process described under 'set-up procedure'.

5. INDUCTION OF TREES FROM BENCHMARK DATA-SETS

The above algorithm for building individual trees (with the pruning

parameter m set to 1) was implemented using a research version of the commercial inductive package XpertRule (Attar Software Ltd 1988). 'Significant candidates' in the description of the algorithm given earlier are those yielding a $2 \times n$ χ^2 exceeding the level corresponding to $P < 0.05$. The implementation was generalized to cater for multi-valued logical attributes and for numerical attributes. The treatment of the latter was essentially that of ACLS (Paterson and Niblett 1982) as also found in CART and C4 (Quinlan 1986), with substitution of expected weights of evidence for entropy criteria. For comparing the Evidencer algorithm with C4, the latter was simulated in XpertRule using the minimum-entropy criterion for attribute selection according to the gain-ratio regime described by Quinlan in the above-cited paper. Six test problems were used, varying in number and types of attribute, number of outcome classes, and size of training set, as shown in Table 2.

Table 2. Characteristics of seven problems used to compare the attribute-selection criteria of Evidencer and C4.

Description of problem	Number of attributes	Type of attributes	Number of classes	Size of training set
Problem 1. Engineering fault diagnosis	20	binary logical	8	8
Problem 2. Autolander	6	mixed logical	2	15
Problem 3. Identifying silhouettes	9	numeric	8	71
Problem 4. Expenses claims	5	logical & numeric	3	37
Problem 5. Credit assessment 1.	20	mixed logical	2	650
Problem 6. Credit assessment 2.	20	mixed	2	1200

The problems were mostly taken from public-domain documents. References are:

Problem 1: A-Razzak, Hassan and Ahmad (1986, 1988). Problem 2: Michie (1986). Problem 3: Shepherd (1983, 1985). Problem 4: Paterson and Niblett (1982). Problems 5 and 6: Al Attar (1989).

6. RESULTS

1. *Domain with binary logical attributes.* A well documented faults table was used. This problem has 20 logical attributes representing diagnostic tests. Each test has the values Y, N or -, and there are eight possible faults. The problem is a good example of the use of rule induction to compress tables (logical reduction). To put it into suitable form for tests involving Evidencer, the domain was split into eight sub-domains, each with the same attributes but only two outcomes: fault1 versus the rest, fault2 versus the rest etc. Trees were produced for each of the eight sub-domains. In every case the Evidencer-produced tree and the C4-produced tree were identical.
2. *Domain with multi-valued logical attributes.* The Autolander problem was used which contains a mixture of binary and multi-valued attributes. Evidencer and C4 produced the same 14-leaf tree.
3. *Domain with numeric attributes.* The task involves identifying the type of chocolate from numeric shape attributes of silhouettes (area, aspect ratio, circularity, etc.). Nine numeric attributes were defined and there were eight classes of chocolate. Again, the domain was split into eight sub-domains, each classifying one type of chocolate against the others. The eight decision trees produced using Evidencer were identical to those produced using C4. Two of the trees are reproduced below.

Problem brandy

```
1  circlty
2  <   78.0:  other  (50)
3  >=  78.0:  area
4             <   115.5:  other  (9)
5             >=  115.5   prornd
6                         <   56.5:  other  (2)
7                         >=  56.5:  bx2ness
8                                    <   76.0:  other  (1)
9                                    >   76.0:  brandy (9)
```

Problem hazel

```
1  asprat
2  <   78.5:  other  (40)
3  >=  78.5:  prornd
4             <   57.5:  boxness
5                        <   64.0:  other  (2)
6                        >=  64.0:  hazel  (8)
7             >=  57.5:  bx2ness
8                        <   76.0:  hazel  (1)
9                        >=  76.0:  other  (20)
```

4. *Domain with both numeric and logical attributes.* Using the 'expense claims' problem, C4 produced a 26-node tree. Evidencer produced an almost identical tree containing 28 nodes, two being 'empty' leaves.

5.–6. *Multivariate domains with noise.* These two domains did not support the induction of decision trees with 100 per cent accuracy, the attributes being insufficient to determine the problem fully. Training sets were constructed by random sampling from files of data acquired in the field. They were thus rather typical of data analysis as encountered in science and industry.

The first credit domain had the outcomes GOOD and BAD credit risk and 20 attributes of credit applicants were used. A class probability tree was extracted from a training set of 650 and tested on a set of 280. The second credit problem was similar but had more examples, 1200 in the training set and 600 to test against. Results from the latter are shown below:

Induction method	*Size of pruned tree*	*Accuracy*
C4	29 nodes	64.6%
Evidencer	22 nodes	66.6%

7. DISCUSSION

As far as concerns comparison of Evidencer's and C4's attribute-selection criteria, performance on the tests showed little to choose between them. In some cases results were identical. In others some small marginal advantages of accuracy or rule-compactness were observed either in one direction or the other. We conclude that the 'expected weight of evidence' criterion satisfactorily solves a long-standing problem attending the entropy-minimization principle. This problem is that the use of the principle tends to bias selection in favour of multi-valued attributes in problems of mixed attribute types. It was in order to patch this that Quinlan introduced the 'gain ratio' adjustment. It can be argued that such patches cannot be other than *ad hoc.* The underlying problem is deeper, and stems from reliance on an undifferentiated conception of entropy as generalized uncertainty, rather than isolating the component of uncertainty which specifically concerns prediction of the outcome class—see Pagallo and Haussler's (1988) discussion of 'mutual information' in this context. In our view this point is fundamental and constitutes strong and sufficient reason to prefer the new criterion to Quinlan's. Further strength is lent by the circumstance that the gain ratio adjustment itself lacks clear justification. Sequential Bayes does not.

In the matter of backward pruning using decision thresholds, a preliminary trial was made in Problem 5. In this problem no use was made of χ^2. In the resultant absence of forward pruning, it was possible to make sufficient test of the θ_1/θ_2 backward pruning scheme to suggest that the Wald sequential cut-off method is safely transferable. Beyond that we do not care to generalize until full implementation of the complete algorithm described in this paper. Comparative trials can then be addressed to industrial-strength applications sufficient to stretch Evidencer's innovative features.

Acknowledgements

This work is indebted to facilities and support from Attar Software Ltd (ASL) of Leigh, Lancs, UK, and from the Turing Institute, Glasgow, UK. By arrangement with ASL, Mr Haider Al Attar modified XpertRule to create the research version described, and ably assisted in its use in the Mathematics Department of the University of Strathclyde to support one of us (DM) in teaching a 4th-year course in AI Data Analysis. We are also grateful to Dr Igor Kononenko for allowing us sight of his writings in advance of publication.

REFERENCES AND BIBLIOGRAPHY

Al Attar, A. (1989). Personal communication.

Angus, J. E. (1989). On the connection between neural network learning and multivariate nonlinear least squares estimation. *Internat. J. Neural Networks.* **1**, pp. 42–6.

A-Razzak, M., Hassan, T., and Ahmad, A. (1986). *ExTran 7 User Manual,* Version 7.2. Intelligent Terminals Ltd (revised 1988), Glasgow.

Attar Software Ltd (1987). *XpertRule* (user manual), Attar Software Ltd, Leigh, Lancs, UK.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth and Brooks/Cole Advanced Books & Software, Monterey, CA.

Carter, C. and Catlett, J. (1987). Assessing credit card applications using machine learning. *IEEE Expert,* **2** No. 3, pp. 71–9.

Duda, R. O., Hart, P. E., and Nilsson, N. J. (1976). Subjective Bayesian methods for rule-based inference systems. *Proc. National Computer Conf.* (AFIPS Conf. Proc. Vol. 45), 1075–82.

van Emden, M.-H. (1974). LIB memo functions. In *POP-2 Program Library documentation.* School of Artificial Intelligence, University of Edinburgh.

Friedman, J. H. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Computers, C-26,* pp. 404–8.

Good, I. J. (1950). *Probability and the Weighing of Evidence.* Griffin, London.

Good, I. J. (1979). *Studies in the history of probability and statistics.* XXXVII A. M. Turing's statistical work in World War II. *Biometrika,* **66**, pp. 393–6.

Good, I. J. (1983). *Good Thinking: the Foundations of Probability and its Applications.* University of Minnesota Press, Minneapolis.

Good, I. J. (1985). Weight of evidence: a brief survey. In *Bayesian Statistics 2* (eds J. M. Bernado, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith). Elsevier (North-Holland).

Good, I. J. (1989). Introductory remarks for the article in *Biometrika*, **66** (1979), "A. M. Turing's statistical work in World War II" (draft 9). Personal communication of a typescript to be published.
Hunt, E. B., Marin, J., and Stone, P. (1966). *Experiments in Induction.* Academic Press, New York.
Michie, D. (1968). 'Memo' functions and machine learning. *Nature*, **218**, pp. 19–22.
Michie, D. (1986). The superarticulacy phenomenon in the context of software manufacture. *Proc. Royal Society (A)*, **405**, pp. 185–212.
Michie, D. (1989). Personal models of rationality. *Turing Institute Research Memorandum -88-035.* Also in *Jour. Statist. Planning and Inference* (Special Issue on Foundations and Philosophy of Probability and Statistics), **21**.
Pagallo, G. and Haussler, D. (1988). Feature discovery in empirical learning. *Research Memorandum UCSC-CRL-88-08*, Computer Research Laboratory, University of California at Santa Cruz.
Paterson, A. and Niblett, T. B. (1982). *ACLS Manual.* Intelligent Terminals Ltd, Edinburgh.
Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo.
Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In *Expert Systems in the Micro-electronic Age* (ed. D. Michie), pp. 168–201. Edinburgh University Press, Edinburgh.
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, **1**, pp. 81–106.
Reiter, J. (1980). *AL/X User Manual*, Intelligent Terminals Ltd, Edinburgh.
Shapiro, A. D. (1987). *Structured Induction in Expert Systems.* Addison-Wesley.
Shapiro, A. D. and Niblett, T. (1982). Automatic induction of classification rules for a chess endgame. *Advances in Computer Chess 3* (ed. M. R. B. Clarke), pp. 73–92. Pergamon, Oxford.
Shepherd, B. (1983). An appraisal of a decision-tree approach to image classification. IJCAI-83, pp. 473–6 (Karlsruhe).
Shepherd, B. (1985). Computer induction versus statistical classifiers in the domain of shape recognition. *M. Phil. Thesis*, University of Edinburgh.
Wald, A. (1947). *Sequential Analysis.* John Wiley & Sons, Inc., New York; Chapman & Hall, London.
Wolfram, S. (1988). *Mathematica: a system for doing mathematics*, Addison-Wesley.

APPENDIX A—KONONENKO'S BAYESIAN NETS

In an article to appear, Kononenko (1989*b*) shows that the 'naïve' Bayes classifier can be naturally implemented in a neural net. In a recent conference contribution (1989*a*) he describes an implementation whereby each iteration of such a net represents a single inference in a chain of inferences. In thus conferring on neural nets some of the properties of an expert system shell, expressive power is lifted above the level of the simple naïve Bayes classifier, reverting to it when iterations are reduced to zero. Some such 'lifting' also results from incorporating the standard Rumelhart and McClelland 'hidden layers' in neural net architectures. The latter results in a scheme for fitting attribute-weights to data which Angus (1989) has shown to be equivalent to a particular way of performing classical least-squares non-linear multivariate

analyses. It would be of interest to know whether Kononenko's method of iterative approximation can be mathematically linked to established multiple regression or other estimation procedures traditionally used by statisticians. In a new paper Kononenko (1989*c*) concedes that the limitations of his system which are due to the independence assumption (see objection (2) in this paper's Introduction) are the same as with 'naïve Bayes'. So severe a limitation certainly does not apply to the non-linear variant of least-squares multivariate estimation discussed by Angus, although it is strictly applicable to the linear variant. Recall that the non-linear variant is equated in expressive power to Rumelhart-McClelland neural nets. It would seem to follow that the latter suffer less under conditions of non-independence than do Kononenko's 'Bayesian networks'.

In describing sequential Bayes as outlined in Michie (1989), Kononenko's new paper contains one or two slips which we take this opportunity to correct. Pruning is said to be defined 'with a threshold', rather than with two thresholds. It is also stated that the algorithms for generation of decision trees cannot be naturally extended to learn incrementally. Incremental modification by re-entry to learning mode is intrinsic to Evidencer's operation, as described in the present paper.

REFERENCES

Kononenko, I. (1989*a*). Interpretation of neural networks decisions. *Proc. IASTED Intern. Conf. Expert Systems and Applications,* Zurich, June 26–28.

Kononenko, I. (1989*b*). Bayesian neural networks. In *Biological Cybernetics* (to appear).

Kononenko, I. (1989*c*). ID3, sequential Bayes, naive Bayes, and Bayesian neural networks. Ljubljana University: Faculty of Electrical and Computer Engineering, unpublished report.

QUALITATIVE REPRESENTATIONS OF KNOWLEDGE

14

Exploring Structures: An Exercise in Model-based Interpretation and Planning

I. Bratko†
The Turing Institute,
Glasgow, UK

Abstract

This paper presents the hardware and software environment of the Freddy 3 Artificial Intelligence based robotics project, which is particularly shaped to support robotic programming in the sense of Brady's definition of robotics as intelligent connection of perception to action. An application of this environment to implementing the StructureMover demonstration is then described in detail. The structure-moving task, accomplished by this demonstration, requires exploration with sensors, and understanding of a structure presented to the robot system. It thus requires the processing of sensory information, model-based interpretation of this information, and planning of both physical actions as well as measurement actions to acquire new information.

1. INTRODUCTION

Brady (1985) defined robotics as 'intelligent connection of perception to action'. Related to this definition and more specifically, the Freddy 3 robotics project sets as its goal the creation of a powerful robot programming environment incorporating the following particular features:

(1) automatic planning;
(2) sensors and model-based processing of sensory information;
(3) machine learning; inductive programming;
(4) multi-robot cooperation and coordination;
(5) human-transparent communication between robots;
(6) various representations of the robot world: logic-based, geometric, naïve physics, deep models, and operational knowledge.

The general architecture to support these features is described in more detail in Mowforth and Bratko (1986). In this paper the hardware

†Current address: E. Kardelj University, Faculty of Electrical Engineering and Computer Science, Ljubljana, Yugoslavia.

and software environment of the Freddy 3 project is presented which aspires to be particularly suitable for programming robots in the Brady sense. Then the StructureMover demonstration is described, which is an application of this environment to implementing a task of moving a structure, presented to the robot, to a new position. As the structure to be moved is not specified, the robot first has to explore it with sensors to understand it. Thus the task involves the processing of sensory information, model-based interpretation of this information, and planning of both physical actions and actions whose purpose is to acquire new information about the structure.

The current Freddy 3 hardware configuration includes the following:

(1) two Unimate Puma 200 robots with LSI 11 controller computers running the VAL II robot programming language (VAL II is a language of the end-effector level);
(2) a Rhino robot controlled directly from a VAX computer;
(3) VAX 11/750 and several Sun workstations networked through the Ethernet;
(4) two cameras sharing a CRS 4000 frame store device;
(5) proximity and beam-break sensors mounted on the fingers of the Puma robots;
(6) a conveyor belt.

Figure 1 shows the software hierarchy for robot programming which spans from the robot arm control to the VAL II language, the low-level sensory information processing in C, to high-level reasoning, planning, and model-based interpretation in PROLOG. Software in the VAX is Unix-based. The implementation of PROLOG is Quintus PROLOG. ROBEYE is a Quintus PROLOG program for communication via Unix pipes between a task-level program and the binary vision system BINEYE, and with another C program, ROBCON, which communicates with the LSI 11 Puma controllers by sending VAL II commands and receiving messages from the controllers and proximity and beam-break sensors. BINEYE (Shepherd 1987) is a simple but practical vision system built around the HIPS set of basic image processing modules. The Turing Institute enhancements and modifications of the basic HIPS package include a learning-from-examples facility, ACLS (Paterson, Niblett, and Shapiro 1982) which enables the user to teach the system to recognize non-overlapping objects by simply showing examples of these objects. Another inductive learning system available for robot programming is RuleMaster (Michie *et al.* 1984) which has been applied to learning plans from examples for block manipulation problems (Dechter and Michie 1984, Shepherd 1985).

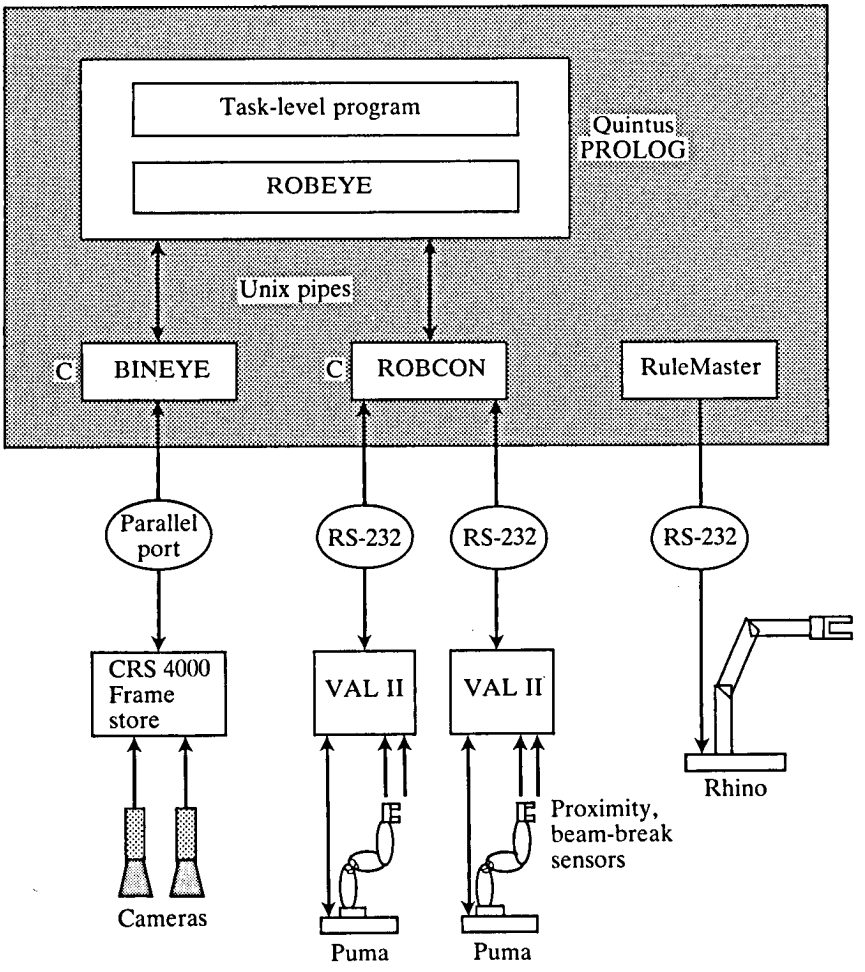


Figure 1. Hardware and software configuration of Freddy 3.

The StructureMover program described below demonstrates the power of using PROLOG in model-based interpretation and planning. The task involves combining three sources of sensor information, as well as planning of physical actions and measurement action. In measurement interpretation, the program makes maximal use of the knowledge of what can be expected in the robot's world. In fact, the point of the exercise is the use of models to compensate for gaps in the sensory information, for example, hidden objects. StructureMover was programmed in PROLOG and integrated with the rest of the Freddy 3 hardware and software environment in two man-months. The structure-moving task of this paper is similar, in respect of planning, to the struture-copying task described in Winston (1972). Model-based interpretation of sensor data

was also investigated by Grimson and Lozano-Perez (1984), although the nature of constraints drawn from the model in their work was different.

2. THE STRUCTURE-MOVING TASK

In short, the task for the robot is:

> Move the structure you see from its current position to the given goal position.

A structure to be moved is made of blocks. Figure 2 shows examples of structures for the moving task. The robot is not told what the structure is. However, it is told what building blocks the structure is made of. For example, for structure 1 in Figure 2, it is told that the structure consists of three 'longblocks' and one 'longslim'. Here, 'longblock' is the user-defined name for a block of dimensions 2*4*2 cm, and 'longslim' is a block of dimensions 2*6*1 cm. The user may help the system by (partially) supplying the orientation of the blocks. For structure 1 in Figure 2, the user may thus specify: 'there are two lying longblocks, one standing longblock, and one lying logslim'.

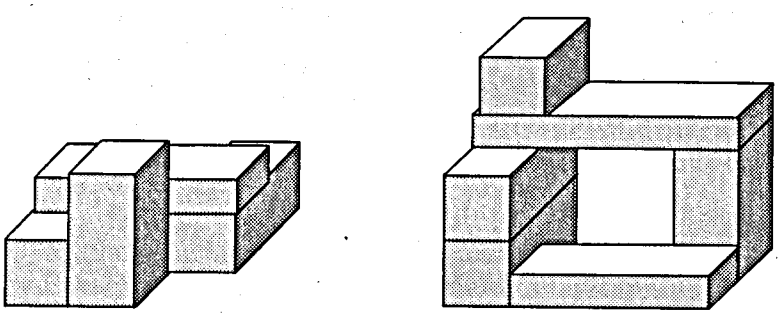


Figure 2. Examples of structures for the moving task.

The robot can manipulate one block at a time. To move the whole structure, it first has to understand it. Therefore, the task consists of exploring, by means of sensory measurements, and disassembling the structure, and then reassembling it at the goal position. For disassembly, the robot may use a set of predefined intermediate locations. Obviously, the exploration (including the disassembly) of the structure is the difficult and interesting part of the task. Since the structure is not known, the disassembly has to be planned carefully, avoiding risky or violent moves. Such moves could disturb blocks in the structure in an uncontrolled way thus resulting in a loss of structural information. No robot action may

ever result in a change whereby the information about the structure would be irreversibly lost.

The repertoire of the system's actions includes the following operations accessible as PROLOG procedures:

pickup(GraspPosition, Presence)

where

GraspPosition=X/Y/Z:Orientation
Presence=object or **no_object**

The effect of the pickup is: Move the robot hand to above the point (Z,Y,zero) (zero is the height of the table) at sufficient height to avoid any object, orientate the gripper according to Orientation, move downwards vertically to height Z, and check for presence of an object between the fingers (beam-break sensor); now grasp if an object is present. The argument Presence will signal whether an object has been grasped or not.

place(GraspPos)

Move the object currently held to a new position GraspPos (that is the position at which the moving object is held at the final position of the move). The trajectory consists of three phases: vertically upwards, then horizontally, and finally vertically downwards until GraspPos is reached.

lookinto(WindowName, Silhouette)

WindowName is the name of a user-defined window, of rectangular shape, into the image array input from the camera (the user can in principle define any number of windows). Silhouette is the binary top-view image of the objects within Window. The silhouette is represented as the union of largest rectangles in the image. The use of windows enables selective processing of various parts of the image thereby saving time.

proximityscan(Area, Height, SwitchPoints)

This causes an investigation by the proximity sensor, mounted on one of the robot's fingers. The robot scans with this sensor at the height Height within Area (again represented as the union of the largest rectangles). The results are returned in SwitchPoints, that is the list of points at which the proximity sensor changed its state (object/no object detected and vice versa), see Figure 3.

3. THE ALGORITHMS

Before looking at the details of the algorithms and their implementation, an example will illustrate the general ideas.

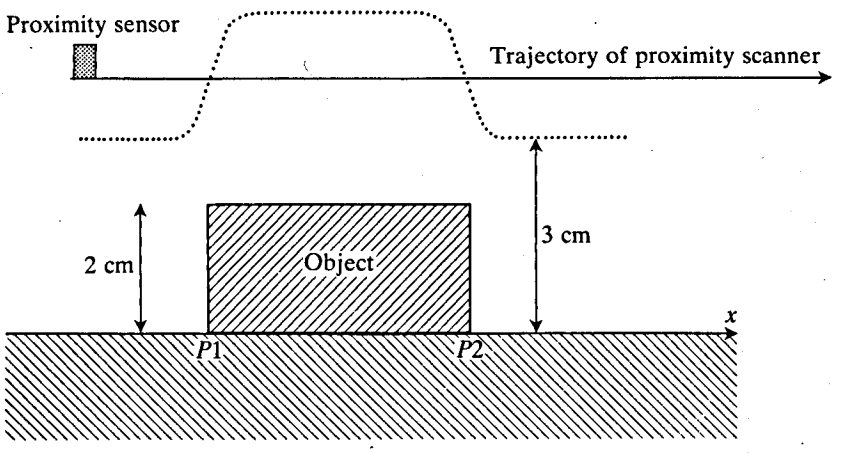


Figure 3. Proximity scanning at height 4 cm above the table. The sensor is pointing downwards. The dotted line indicates the region where the sensor will detect the proximity of the object when scanning is in the x direction within the area above the object. The signal will go on at point $P1$ and off at $P2$. This information is represented as a PROLOG list of the form: Switchpoints = [$P1$: onX, P2: offX].

Let the structure to be moved be the one in Figure 4. Here is the trace of the system's behaviour when solving this problem.

1. The system is started by being told that there are the following blocks in the structure:

   2 * standing longblock
   1 * lying longblock
   1 * lying longslim

2. The top view silhouette of the structure is extracted from the BINEYE vision system. This silhouette is reported as a rectangle of the length 8 cm and width 2 cm.

3. This silhouette is interpreted with respect to the specified material. The alternative candidate structures that satisfy the known constraints, together with the general physical constraints (e.g. stability), are found. These are in this case represented by a list of 32 structured PROLOG objects.

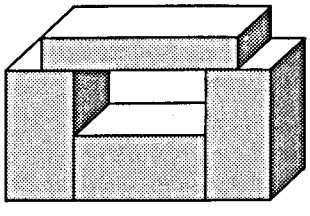


Figure 4. Structure for detailed example.

4. The candidate configurations are of various heights. The tallest candidates are considered first; in the case that the actual structure is lower, actions attempted at the greatest height will not damage the actual structure. In our case, the maximum height is 8 cm. There are two groups of candidate structures of this height, illustrated in Figure 5. The system now tries to find a grasping position suitable for all the candidate structures of height 8 cm. There is no such *common* grasping position in our case. As there is no common grasping position the system tries to find at least a *safe* grasping position. Fortunately, the two candidate grasping positions at height 7 cm, labelled G1 and G2 in Figure 5, are safe in the sense that grasping at any of them will not harm the structure should the corresponding hypothesis be false. Thus G1 is chosen and the action performed is

   pickup(G1, Presence)

   Since there is no object at G1, the information returned by the pickup procedures is Presence = no_object. The pickup action has thus ended as a failed grasping experiment.

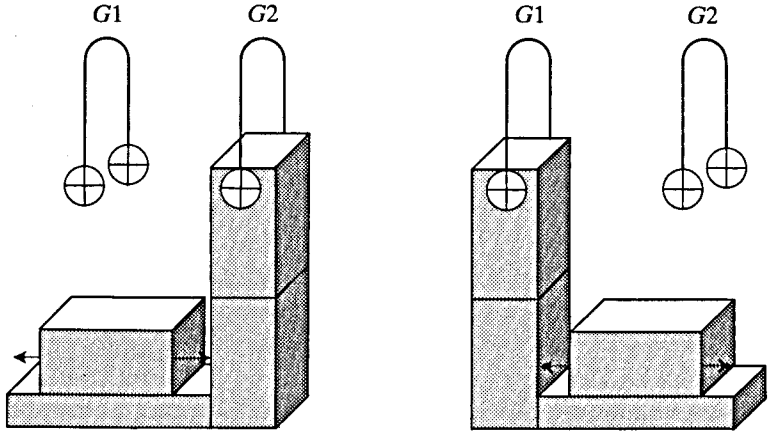


Figure 5. Candidate structures of height 8 cm, G1 and G2 are safe grasping positions. The arrows indicate the ranges of possible positions of the lying longblock.

5. The candidate structures that are not compatible with the new constraint that there is no object at point G1 must be eliminated. This rules out the left-hand side family of structures in Figure 5. There is now a common grasping position, G2, common to all the remaining candidate structures of height 8 cm. The action

   pickup(G2, Presence)

   again fails to grasp anything. The candidate structures incompatible

with this result are eliminated again, ruling out all the candidate interpretations of height 8 cm,

6. The maximum height of the remaining candidate structures is 7 cm. There is a large number of candidate configurations of this height. They do not share a common grasping position, and furthermore, there is no safe grasping position. Therefore the action taken is

   proximityscan(Area, 7, SwitchPoints)

   where Area is the area of the structure's silhouette. Since the actual structure is only 5 cm tall, this measurement returns the empty list of switch points:

   SwitchPoints = [ ]

   As a result, all the candidate interpretations of height 7 cm are eliminated from consideration.

7. At this point, there are five possible configurations left consistent with the information known so far and with the other constraints. They are shown in Figure 6. They all have a height equal to 5 cm. There is a safe grasping position G3, and the action

   pickup(G3, Presence)

   is performed. This time the grasping succeeds (Presence = object), although the robot does not know yet what object it is holding. It could be the long slimblock or a longblock which corresponds to the structures 1 and 2 of Figure 6. These two structures remain as the possible candidates whereas the other three structures in the Figure are discarded.

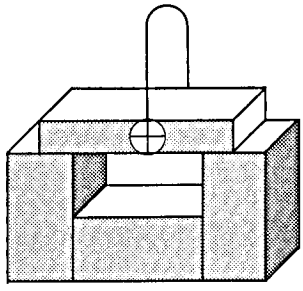
8. The next action is:

   place(free_location_1)

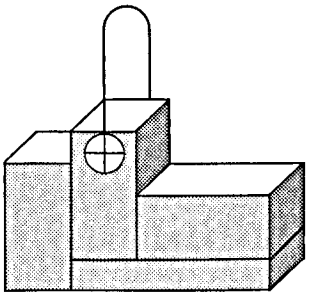
   The long slimblock is placed into the first free location. There is a small difficulty associated with this action since the robot does not know which object is being placed: either the lying long slimblock or a standing longblock. Therefore it is not quite clear at what height the object should be released. There is a choice between risking to drop the long slimblock from excess height, and pressing the standing longblock against the table. Fortunately, the geometry of the robot's fingers allows a block to slip smoothly along the fingers if pressed against the table, therefore the second alternative is chosen.

9. The object that has been just placed is looked at:

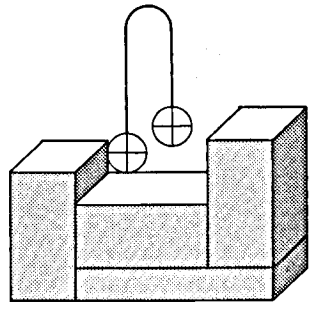
   lookinto(free_location_1, Shape)



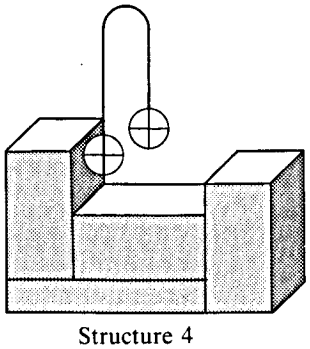
Structure 1



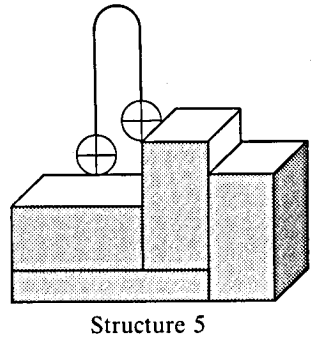
Structure 2



Structure 3



Structure 4



Structure 5

Figure 6. Candidate structures of height 5 cm. There is a safe grasping position.

The silhouette returned in our case is a rectangle of dimensions 2*6 cm. This last piece of information is sufficient to identify the block just moved, which renders the structure finally disambiguated.

10. Since the structure is now completely known, the rest of the disassembly process is smooth and easy. The disassembly is followed by the reassembly at the goal location, in the reverse order of the disassembly process.

Figure 7 shows the top-level code of a PROLOG implementation of this

process. For reasons of clarity, the original program was slightly simplified for this presentation. The omissions from the original deal with the user interaction and various possible inconsistencies in the robot environment. The Edinburgh syntax conventions of PROLOG are used (e.g. Bratko 1986, or Clocksin and Mellish 1981), where names beginning with a lower case letter denote atoms (i.e. constants), and those with upper case letters denote variables. It is clear from the code that structures are decomposed from top to bottom, and grasping experiments are preferred to proximity measurements.

```
% Top level procedure: go( Blocks_in_scene)

go( Blocks) :-
  lookinto( startwindow, Silhouette),
  interpret( Silhouette, Blocks, Structures),
  disassemble( Structures, ActualStructure),
  reassemble( ActualStructure)

% disassemble( CandidateStructures, ActualStructure)

disassemble( Structures, Structures) :-
  all_blocks_moved( Structures).      % Whole struct. disassembled

disassemble( Structures, ActualStruct) :-
  maxheight( Structures, Height),          % Maximum height
  separate( Structures, Height, Tall, Low), % Split into low and tall
  removeblock( Tall, Outcome, NewStructures),
  continue( Outcome, NewStructures, Low, ActualStruct).

% Outcome = moved (block successfully remove) or not_moved

continue( moved, NewStructs, _, ActualStruct) :-
  disassemble( NewStructs, ActualStruct).

continue( not_moved, [ ], LowStructs, ActualStruct) :-
  disassemble( LowStructs, ActualStruct).

continue( not_moved, TallStructs, LowStructs, ActualStruct) :-
  non_empty( TallStructs),
  top( TallStructs, Height, Area), % Top area of TallStructs at Height
  proximityscan( Height, Area, SwitchPoints),
  consistent( TallStructs, SwitchPoints, ConsistentStructs),
  removeblock( ConsistentStructs, Outcome, NewStructs),
  continue( Outcome, NewStructs, LowStructs, ActualStruct).

% Removing a block from the structure to an intermediate place
% removeblock( CandidateStructs, Outcome, NewCandidateStructs)
```

```
removeblock( [ ], not_moved, [ ]) :- !.        % No candidate structure

removeblock( Structs, Outcome, NewStructs) :-
  grasppos( Structs, GraspPos),
  pickup( GraspPos, Presence),          % Object grasped?
  continue_move( Presence, Structs, GraspPos, Outcome, NewStructs).

grasppos( Structs, GraspPos) :-
  commongrasp( Structs, GraspPos), !;  % A common grasp position?
  safegrasp( Structs, GraspPos).        % A safe grasp position?

continue_move( object, Structs, GraspPos, moved, NewStructs) :-
  getplace( FreeLocation),
  place( FreeLocation, GraspPos),
  lookinto( FreeLocation, TopView),
  update( Structs, GraspPos, TopView, NewStructs).

continue_move( no_object, Structs, GraspPos, Outcome, NewStructs) :-
  freepos(GraspPos, Structs, TrueStructs), %GraspPos free in TrueStruct
  removeblock( TrueStructs, Outcome, NewStructs).
```

Figure 7. The StructureMover program: top level procedures in PROLOG.

In the following paragraphs we look more closely at some of the important procedures: interpret, commongrasp, safegrasp.

3.1. interpret(Silhouette, Blocks, Structures)

Interpret finds Structures as possible interpretations of the given Silhouette, under the specified building material Blocks and other physical constraints. StructureMover makes several strong assumptions with respect to the structure that can be expected in the scene. Briefly, it expects a rather regular, 'engineering-type' structure in which all the blocks are aligned with the *x*-*y*-*z* axes. Essentially, the interpretation problem is a kind of bin-packing problem where the given blocks are to be packed into a given volume (determined by Silhouette) under stability constraints. The interpretation program assumes that any structure presented to the robot is 'superstable'. A structure is called superstable if all the bottom corners of any block in the structure are supported. Accordingly, stacks and arches are superstable, whereas seesaw-type structures are not. The major problem in implementing the interpret procedure is that of combinatorial complexity, since the complexity of interpretation grows exponentially with the number of blocks in the structure. The program uses certain time-saving heuristics, including these two principles:

(1) fill the given volume in the order from the bottom to the top;

(2) fill the convex corners of the silhouette first (this is justified by the superstability constraint).

3.2. commongrasp(Structures, GraspPosition)

GraspPosition is a grasping position (position and orientation) shared by all the configurations of blocks represented by Structures. There has to be enough space for the robot fingers to prevent collision with any of the blocks in any of the candidate structures. Figure 8 illustrates how a common grasping position can be determined. The problem of determining a common grasping position is again computationally complex because of its combinatorial nature with respect to the number of candidate structures and number of blocks in the structures. One heuristic used in the StructureMover program is to consider only the topmost blocks in the candidate structures.

3.3 safegrasp(Structures, GraspPosition)

Here GraspPosition is a grasping position in at least one of the candidate

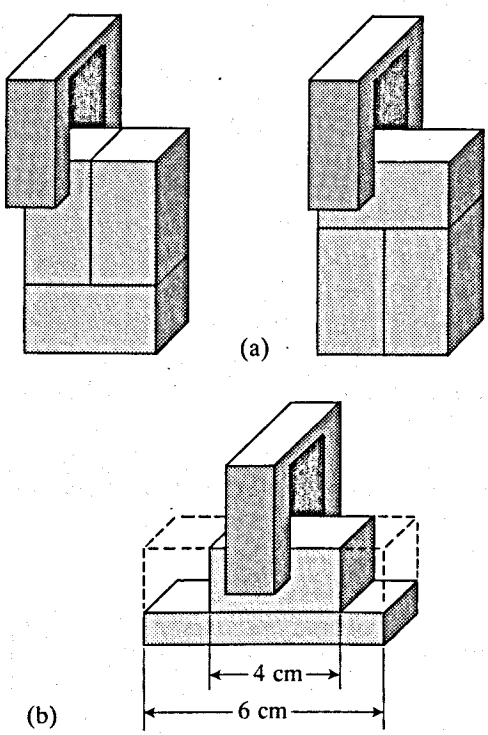


Figure 8. Examples of common grasping positions. (a) The position of the fingers indicated is the common grasping position for both strutures. (b) Longblock on top of longslim block; even if the displacement of the longblock is not known there is a common grasping position as indicated.

configurations in Structures, and is either a grasping position or is safe with respect to any other configuration. Of course, there are situations in which there is no safe grasping position. In such cases proximity measurements are necessary, as illustrated in Figure 9. Again, the safe-grasp relation is computationally complex.

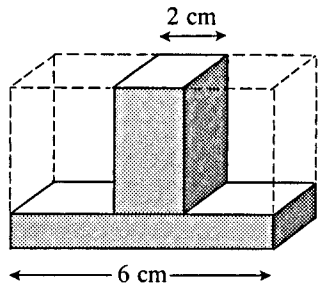


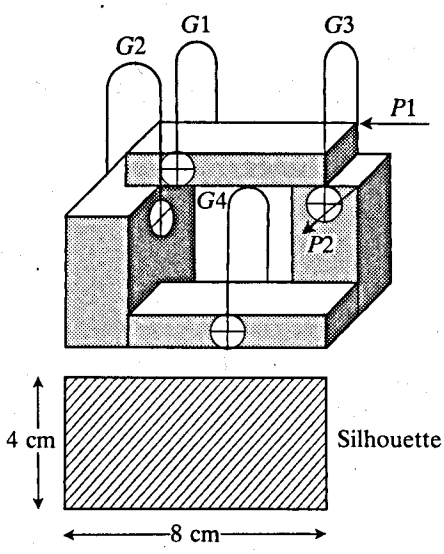
Figure 9. A standing longblock on a longslim block. For the set of structures with various displacements of the longblock with respect to the longslim block, there is no safe grasping position.

Several computational functions depend on the representation of free space or free area. In StructureMover, these are represented as the union of all the largest boxes or rectangles enclosed in the free space or free area respectively. This representation facilitates the testing for collisions between blocks, or for verifying that there is sufficient space for placing a block or for the robot fingers when determining a grasping position.

4. PERFORMANCE AND LIMITATIONS

The difficulty of the moving task depends critically on the structural complexity of the structure to be moved. If the blocks are separated or the structure consists of stacks or similar simple constructs then the interpretation task is easy even for a large number of blocks in the scene. In the case of compact structures with overlapping blocks the number of possible interpretations grows fast with the number of blocks. In general, the system can easily handle complex structures of up to five blocks, whereas the present StructureMover implementation may become rather slow with a higher number of blocks because of the combinatorial nature of several subtasks associated with the exploration of structures. For illustration of the program's performance, Figures 10 and 11 give two more examples of structures and corresponding solution traces. The real time needed to disassemble and move structures like these is typically in the order of one or a few minutes. Typical breakdown of this is roughly as follows:

25% low-level vision done by the BINEYE system (grabbing picture,



Told: 2 * longslim/lying; 1 * longblock/standing; 1 * squareblock/standing

There are 40 candidate configurations. Maximum height=8 cm

Proximity scan at height 8 cm returns []. Number of remaining candidate configurations=34. Maximum height=5 cm.

Proximity scan at height 5 cm returns [*P*1: on*X*]. Number of remaining candidate configurations becomes 3.

Grasp at *G*1, object grasped. Move the object and look at it. There are 2 configurations left.

Proximity scan at height 4 cm returns [*P*2: off*Y*]. Now there is only one candidate structure left.

Grasp at *G*2, move object. Grasp at *G*3, move object. Grasp at *G*4, move object. Reassemble.

Figure 10. A structure and the trace of exploring it. *G*1 to *G*4 are grasping positions, *P*1 and *P*2 are proximity switch points detected.

smoothing, and noise reduction, computing parameters of binary blobs detected in the image);

25% high-level, model-based reasoning in PROLOG (interpretation of binary blobs returned by low-level vision and other sensory information, planning of further measurements, planning of grasping actions);

50% actual physical manipulations performed by the robot arm.

The percentages above, of course, vary significantly with the complexity of the structure. For more complex structures, the time necessary for the low-level vision will remain similar in absolute terms whereas the high-level reasoning will expand.

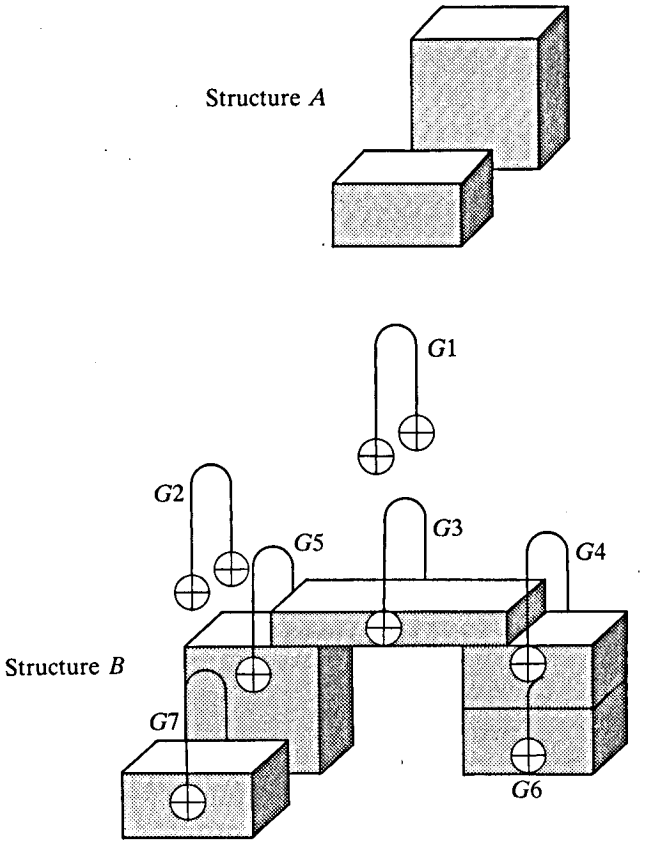


Figure 11. Structure A cannot be solved without the use of the proximity sensor (no safe grasping position). It is interesting that adding more blocks to obtain structure B helps, so that structure B can be solved without the proximity sensor. The reason is that the system will know that the lying longslim block must be supported at both ends, and therefore the square block must be behind the isolated longblock. The grasping sequence $G1, \ldots, G7$ is found without any proximity measurement.

Apart from the combinatorial complexity problems, there are several other limiting factors in the present implementation.

1. *Constraints with respect to structures:*

(a) blocks have to be aligned with the x–y grid;

(b) structures must be 'superstable';

(c) resolution in displacement of blocks in the model is assumed to be 1 cm, although actual structures with finer displacement are still normally handled correctly even if the actual grasping position relative to the object grasped does not exactly correspond to the position in the system's model.

2. *Limitations with respect to the gripper* The simple gripper used in our case allows only objects about 2 cm wide to be grasped. Additional constraints are imposed by the optic fibres attached to the fingers and used by the proximity and beam-break sensors which makes the whole hand rather awkward to manipulate. Thus, for example, two 'longblocks' (2*4*2 cm) aligned in parallel at a distance of 3 cm, say, cannot be grasped at all. Such structures therefore cannot be manipulated unless tricky moves are employed, e.g. pushing objects.
3. *Limitations due to inaccuracy in measurements* Gross approximations of the visual and proximity information are necessary; these approximations are justified by the use of the strongly constrained models of the possible structures.

5. CONCLUDING REMARKS

The work can be naturally extended in many interesting ways. Some of them belong to major research problems of AI-based robotics. Here are some possibilities:

1. Introducing composite objects as illustrated in Figure 12. More interesting objects can be obtained by simply sticking the primitive block-type objects together. Most of the existing high-level reasoning in StructureMover would be sufficient to handle objects composed in this way.

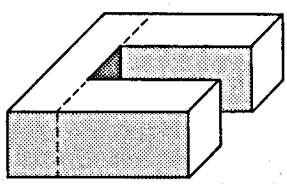


Figure 12. An example of extending the blocks world to more interesting objects using blocks as primitives.

2. Relaxing the superstability constraint to the normal stability.
3. Systematic treatment of measurement errors.
4. Error recovery in plan execution.
5. Planning of tricky moves, for example those that involve the pushing of objects, (a traditional STRIPS-style planner like WARPLAN [Warren 1974] could be used to compile a repertoire of stereotype macro moves to be used by StructureMover to achieve special effects).
6. Naïve physics reasoning and qualitative modelling of the blocks world (Figure 13 illustrates motivations behind such a facility).

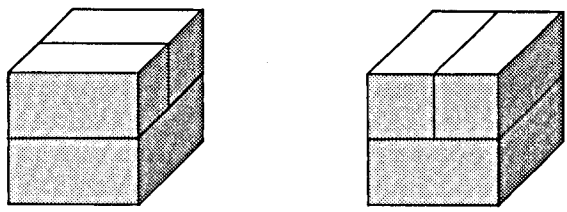


Figure 13. How can the system disambiguate between these two candidate structures? Sensors do not help in this case and 'tricky' moves such as pushing are necessary. One strategy to destroy the structure, without loss of information, would be to use a longslim block as a tool and push both top blocks, first towards the right and then forward, by 1.5 cm each time. In the case of structure *A*, a longblock will finish on the table in front of the other blocks; in the case of structure *B*, a longblock will finish to the right of the other blocks.
We do not have to know the exact location of this longblock, but we have to know it approximately or qualitatively. Mechanical synthesis of such recognition strategies can thus be based on qualitative physics.

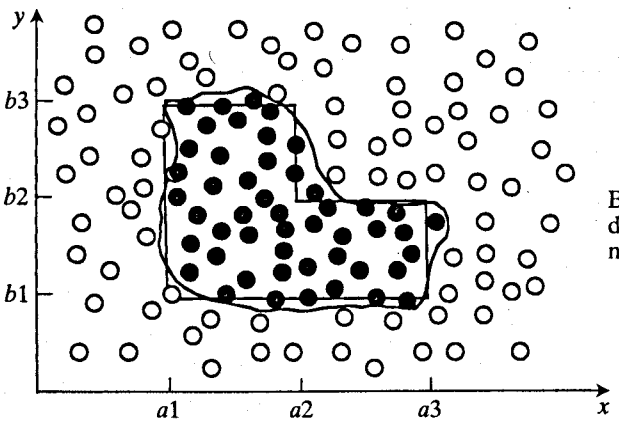
7. Optimization of plans, both in respect of physical actions performed by the robot arm, and the planned measurements according to the expected information gain.

From the experience obtained in this study, it seems a good idea to keep the numerical world that talks to the sensors clearly separated from the high-level relational representation, where the logical reasoning is done. If a mixture of both representations is used, then the clarity of the logical reasoning is marred and much of the advantage of logic thereby lost. In PROLOG, for example, mixing both worlds can lead to unexpected failure in the unification process due to a small mismatch caused by numerical errors in real numbers. A clearcut, although possibly non-deterministic interface between the numerical representation and the logic-based representation would seem to be a good idea.

One way of transforming a numerical representation, marred with noise, such as a binary image, into a logical description is through the use of an inductive learning facility. Figure 14 illustrates how a decision tree generating inductive program of the ID3 type (e.g. Quinlan 1983), with added tree-pruning mechanisms as in the Assistant program (Cestnik, Kononenko and Bratko, 1987) can be used.

Finally a question can be asked: What is the point of such exercises, apart from demonstrating that model-based interpretation may compensate for lack of more sophisticated sensors, as in our case, for example, a very simple binary vision system with a little help from other simple sensors is adequate? Is there a *practical* situation where a similar approach can be applied?

It is not hard to imagine a robot system with a sophisticated user-interaction facility for the naïve user. Such a user would be supposed to



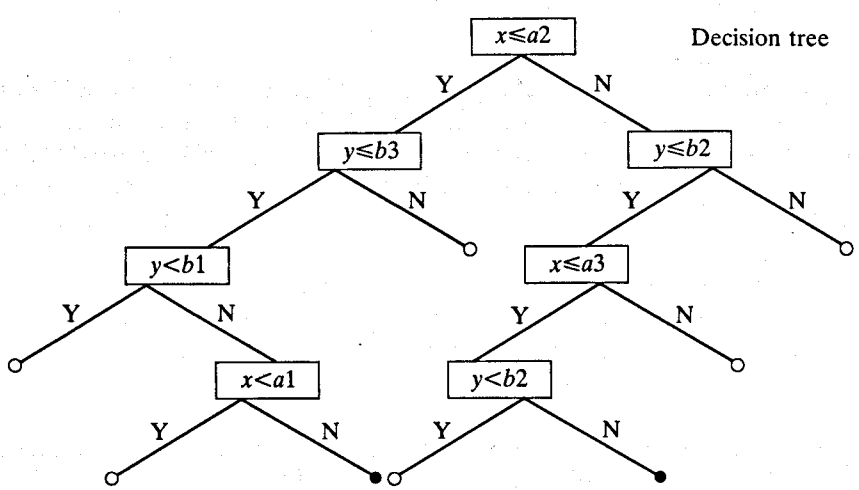


Figure 14. Inductive learning approach to extracting compact descriptions from noisy images. The decision tree built by a pruning inductive tree generator represents a rectangular polygon.

instruct the robot, without any programming knowledge or any reference to geometrical descriptions. How would such a user specify structure 2 of Figure 2, for example? For a complete specification, the user would perhaps have to make reference to a coordinate system, make sure that the robot understands that same coordinate system, and then give exact coordinates of the objects in the scene, or alternatively he or

she would use relations such as 'aligned with' or 'supported-by', or 'behind' from a particular point of view. All this requires a certain effort and degree of sophistication on the part of the user and involves the risk of ambiguous description. It is straightforward, however, just to specify what objects there are in the scene, and possibly to specify some easy relations, such as 'lying' or 'standing'. Using this partial information as constraints, a system like StructureMover would then identify the actual situation through measurements and constraint-based interpretation.

REFERENCES

Brady, M. (1985). Artificial Intelligence and Robotics. *Artificial Intelligence*, **26** (No. 1), pp. 79–121.

Bratko, I. (1986). *Prolog Programming for Artificial Intelligence.* Addison-Wesley.

Cestnik, B., Kononenko, I., and Bratko, I. (1987). ASSISTANT 86: a knowledge-elicitation tool for sophisticated users. In *Progress in machine learning* (ed. I. Bratko and N. Lavrač). Sigma Press, Wilmslow, UK.

Clocksin, F. W. and Mellish, C. S. (1981). *Programming in Prolog.* Springer Verlag.

Dechter, R., Michie, D. (1984). *Induction of Plans.* The Turing Institute, Glasgow (TIRM-84-006).

Grimson, W. E. L. and Lozano-Perez, T. (1984). Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, **3**, pp. 3–35.

Michie, D., Muggleton, S., Riese, C., and Zubrick, S. (1984). RuleMaster: a second-generation knowledge-engineering facility. *First Conf. on Artificial Intelligence Applications,* Denver, 5–7 December 1984. Silver Spring, Md.: IEEE Computer Society, pp. 591–7.

Mowforth, P., and Bratko, I. (1987). AI and Robotics: flexibility and integration. *Robotica.* **5**, 93–8.

Paterson, A., Niblett, T., and Shapiro, A. (1982). *ACLS User Manual.* Glasgow: Intelligent Terminals Ltd.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. *Machine Learning: An Artificial Intelligence Approach* (ed. R. S. Michalski, J. G. Carbonnell and T. M. Mitchell) Kaufmann.

Shepherd, B. (1987). BINEYE: a flexible experimental robot vision system incorporating inductive learning. The Turing Institute, Glasgow.

Shepherd, B. (1985). GENARCH: a practical solution to a general arch building problem using RuleMaster. *Freddy 3 Project: Year 1* (ed. I. Bratko) The Turing Institute, Glasgow.

Winston, P. H. (1972). The MIT robot. *Machine Intelligence 7* (ed. B. Meltzer, D. Michie) Edinburgh University Press.

Warren, D. H. D. (1974). *WARPLAN: a system for generating plans.* University of Edinburgh: Department of Computational Logic (Memo No. 76).

15

Learning of Causality by a Robot

P. H. Mowforth
The Turing Institute and the University of Strathclyde,
Glasgow, UK

T. Zrimec
E. Kardelj University,
Ljubljana, Yugoslavia

Abstract

This paper describes an experiment in which a robot is allowed randomly to explore the domain of object-pushing via controlled experiments. The experiments consist of recording signals from sensors before and after an action has taken place. Each experiment may be considered as a state transformation recorded as a sequence of attribute values. Treating each transformation as a training example, a large set of data was collected and subjected to rule induction. Robust and useful transformations were discovered which were represented as a hierarchical qualitative model. Further, results show that an actor-oriented, co-ordinate frame provides the most compact description for the problem. One final observation is that this style of experimentation offers the potential for closed-loop learning in that, unlike other domains, as far as the robot is concerned, its world is the oracle.

1. INTRODUCTION

Robots are typically difficult to program. Within this context we can consider three approaches that may be taken towards the goal of automating the task. The first approach follows from a high-level task specification, the second from the use of sensors, and the third from allowing the robot to learn as a supplement to predefined behaviour.

The first approach is via the automatic synthesis of robot control programs from a higher-level programming environment. This may include the development of systems for the automatic synthesis of plans. These systems, often referred to as planners, use various forms of constrained search to find a sequence of operations which will transform a given initial problem into a goal. Examples of such systems are GPS [8], Graph Traverser [3], STRIPS [5], and NONLIN [11]. One of the authors has developed such a system [12] (see Figure 1), which has already been used to control an assembly robot for industrial applications. Indeed,

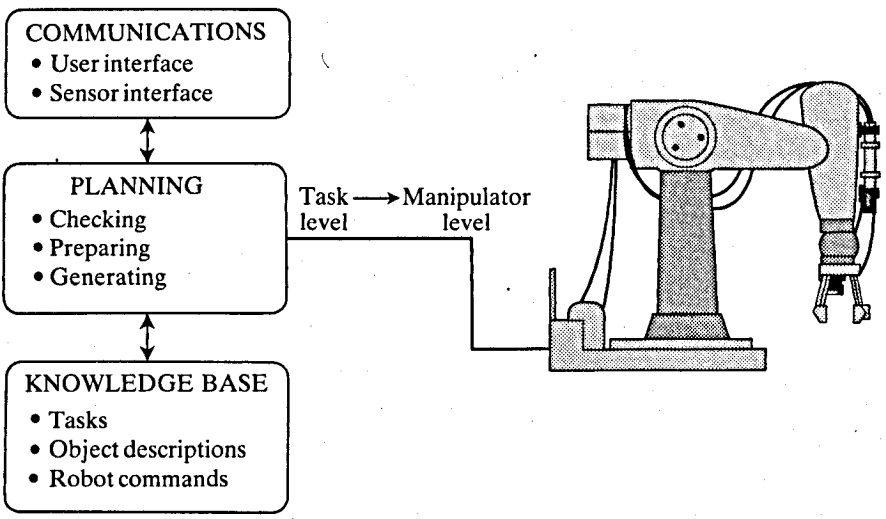


Figure 1. Block diagram of the ROPRO system used to automatically generate manipulator-level code from a PROLOG specification.

part of the motivation for the research described in this paper is a direct consequence of appreciating the difficulty and tedium of developing systems in which a complete framework of specification must be provided.

A second approach to the automation process is via the use of sensory information. Here, a control program can leave certain information in the form of variables whose values are provided by a sensor rather than by direct calculation. For example, rather than calculate the exact position and orientation of an object from some pre-stored knowledge of the problem, variables such as *position* and *orientation* can be provided whose values can be set automatically from sensory systems. Although this offers much potential, a distinction needs to be made between having a sensory signal and knowing how to make effective use of that signal. Having a camera does not mean that you have vison. The knowledge that relates a sensory variable to achieving some step in the control program still needs to be coded by a programmer.

The third approach to the problem involves getting the robot to learn. While this approach is clearly tricky, in principle it offers the greatest potential for automation. One early related piece of work by Michie and Chambers [6] involved a simulation of a pole and cart (a two degree of freedom mobile robot) for which a simple credit assignment algorithm slowly learned to balance the pole. This was achieved by reinforcing condition-action pairs which directly contributed to pole balancing combined with an exploration heuristic designed to ensure the avoidance of local minima. A later piece of related work by Dufay and Latombe [4],

used a two-phase approach to building robot programs: a training phase which produced a number of execution traces and an induction phase which transformed the traces into an executable program. The task was to insert a pin into a chamfered hole, a task which, due to geometric uncertainty, involved the planner in generating several different sequences of motion which were then subjected to iterative transformations such as the merging of nodes and arcs labelled by motions and states regarded as equivalent by rewriting rules. One final related project described by Dechter and Michie [2] involved the use of an inductive learning algorithm which was used in the construction of a robot plan for arch building.

The factors which contribute to the current experiment are based on a few simple observations and preconceptions:

1. First, if we are to fill in the gaps in a poor or incomplete task specification we will need to make use of broadly generalizable knowledge.
2. Such knowledge should be learned on the basis of results gathered in the real world rather than under simulation.
3. A random number generator is a very easy (and natural) way to vary interaction with the world so allowing a broad exploration of a problem domain.
4. The only way the robot can describe its world is in terms of sensory signals. Hence, any model produced by learning should be expressed directly in terms of the natural signals produced by sensors rather than in terms of, for example, abstract geometry.

Within this framework a sequence of experiments was conducted with the goal of exploring the problem domain of object pushing. The aim is to let the system discover how its world works through play and random exploration. To achieve this we must first develop an algorithm which will allow the robot to learn the relationship between action and perception on the basis of performing experiments in its world.

2. METHOD

The experiments described here were conducted using the Turing Institute's advanced robotics research test-bed, [7]. The Freddy 3 environment currently consists of two Puma 260 robot arms, vision systems, speech systems, and other sensors each of which is controlled, in real time, from Quintus PROLOG running on a Unix network. The current experiment uses one robot, a vision system, a speech synthesizer and a PROLOG control process (see Figure 2).

The PROLOG process for robot control (ROBEYE) communicates with a

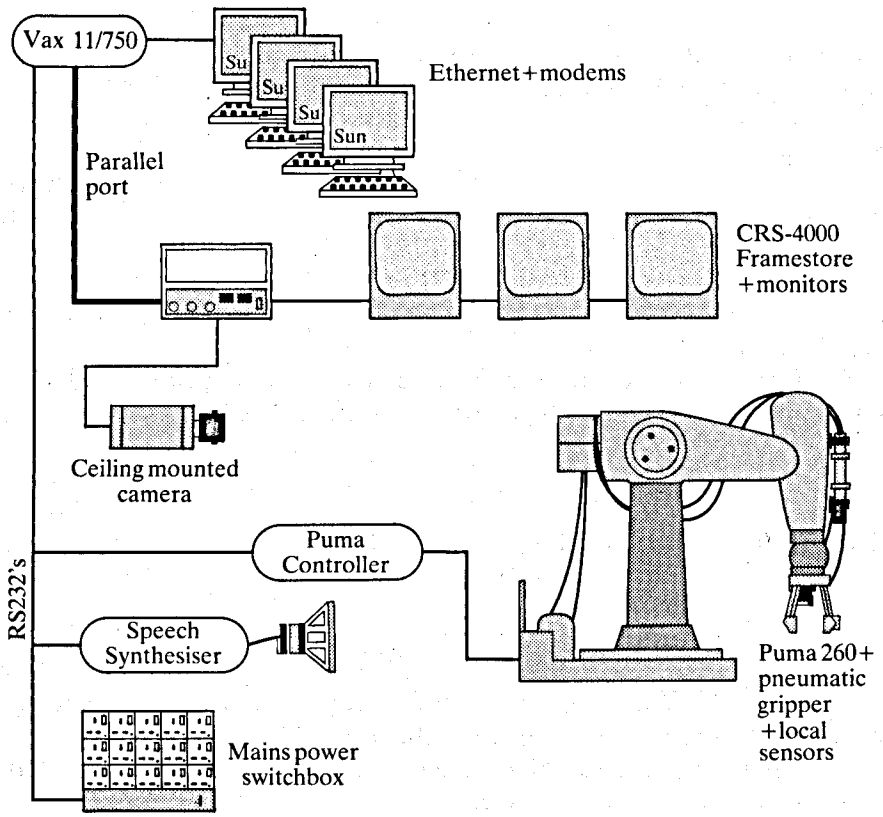


Figure 2. The hardware configuration for the equipment used in the experiment.

vision system process (BINEYE) and other sensory processes through which it gets information about its world. For example, the vision system performs object segmentation via thresholding in the specified window and recognition by using inductively generated rules, so providing information about object class (symbolic name), object position (centroid x,y) and object orientation (principal axis). Examples of vision windows can be seen in Figure 3a.

The object-pushing experiment required a window in the robot working area in which (a) the vision system operated, and (b) there were no robot singularities. The window was a square on the table surface of dimensions 10 cm × 10 cm which was named play_window (the larger of the two windows in Figure 3a). There were a few additional windows which had to be processed in order to calibrate the experimental environment. This was achieved by defining a common coordinate system for the robot and the vision system. The smaller window in Figure 3 shows one of the two calibration crosses used to achieve this.

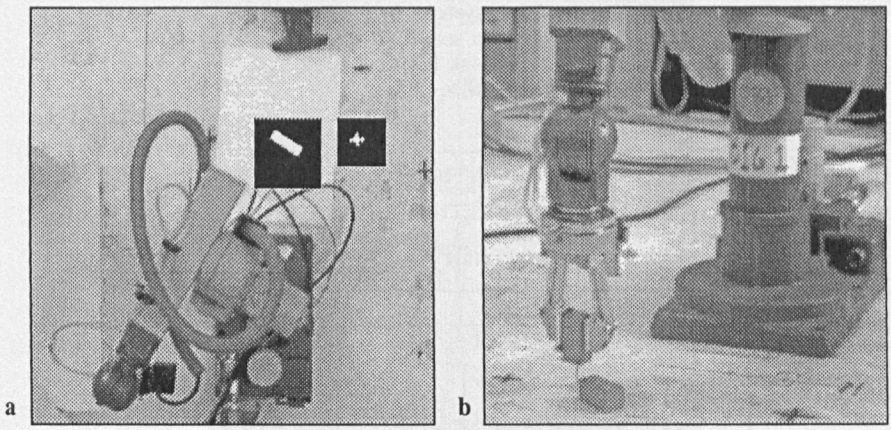


Figure 3. Two views of the experiment, one overhead from the vision system showing two windows and the other from the side (digitized output).

The robot hand held a pushing tool (a pen) which was picked up from the tool_window using BINEYE at the start of the experiment. Next, the robot asked, using its speech synthesis unit, for an object to play with. A human then placed a wooden block, randomly oriented, within the play_window. The vision system then noted the centroid start position (X_obj_start, Y_obj_start) of the object as well as its principal axis which defines the object orientation (Angle_obj_start). The experiment was controlled by a PROLOG program which generated two random numbers for the robot start position on the boundary of the play_window (X_rob_start, Y_rob_start). Two more random numbers were generated for the ending point of the action vector (X_rob_end, Y_rob_end), again on the play_window boundary. The robot then slowly swept the pen through the play_window along the line of the action vector. BINEYE then looked into the play_window and recorded the ending position of the object centroid (X_obj_end, Y_obj_end) as well as its final transformed orientation (Angle_obj_end). Figure 3 shows two views taken at various stages of the experiment; in Figure 3a we show the view as seen by the overhead camera looking down on the robt while in Figure 3b we see a side view. Figure 4 shows a block schematic illustrating the various stages of the experimental method.

Because of the inherent symmetry of the block, BINEYE can provide orientation information only over a range of 90 degrees. To provide 360 degrees of orientation, a black spot was painted on one end of the block and a separate threshold used to segment it. Combining both the silhouette of the object with the silhouette of the spot we were able to define real orientation uniquely. Occasionally the block was pushed out of the play_window. When this happens, ROBEYE automatically invokes

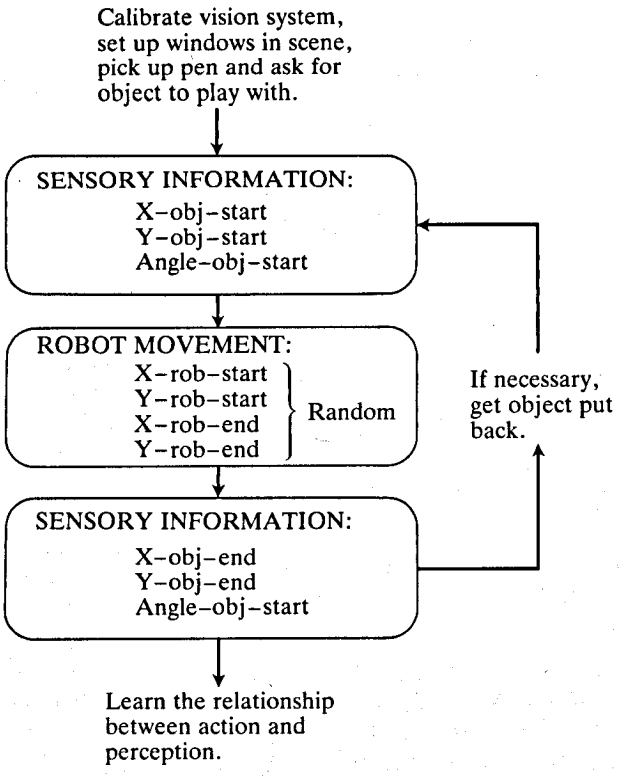


Figure 4. A block schematic for the experimental method.

a larger window and, before asking for human help, records the final position and orientation of the block.

The robot carried out 106 experiments with a wooden block of dimensions 2cm × 2cm × 4cm and the information from each experiment was written on a file. At a later stage (see Section 4 on model generation), 157 further experiments were carried out using the same block. Figure 5 shows a number of random action vectors carried out by the robot. The lines were drawn by the robot using its pen and demonstrate that the random number generator allowed a broad exploration of the problem space.

3. GETTING THE COORDINATE FRAME RIGHT

The next step following data collection was to discover some general rules for the domain using the method of '*learning from examples*'. We used the learning system Assistant:86 [1] which is similar to Quinlan's ID3 algorithm [10]. This algorithm has the ability to induce a general description of concepts (classes) from examples of these concepts

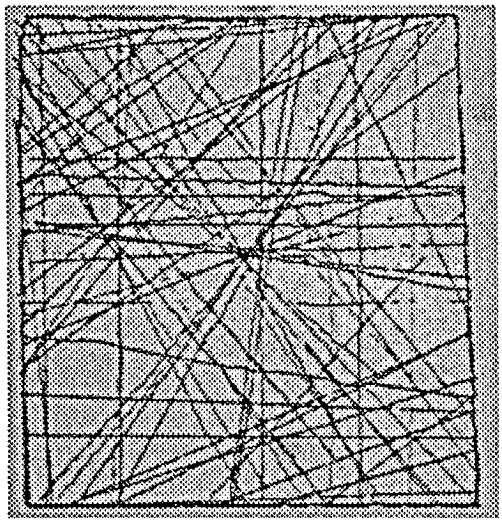


Figure 5. Lines drawn by the robot during the experiments (digitized output). Results show that the random number generator allowed a broad exploration of the problem space.

(classes) and to produce a symbolic description of these concepts in the form of a decision tree. Examples are objects of a known class described in terms of attributes along with their values. Our data consists of ten real numbers from which we had to prepare learning examples—attributes and class information. In order to define classes we must first generate a goal. As the goal is to understand what causes a change in perception, our class values were related to perceptual change. In this way we got three raw classes; *DX* (change in *X* position of the object), *DY* (change in *Y* position of the object) and *DA* (change in orientation of the object). The other seven numbers (*X*_obj_start, Y_obj_start, Angle_obj_start, *X*_rob_start, *Y*_rob_stgart, *X*_rob_end, *Y*_rob_end) were used as raw attributes.

In the first learning experiment we made a simplification with two classes:

change = 0, and
change ≠ 0,
(change stands for object change in either position and orientation).

The induced tree was very large with knowledge expressed in terms of values strongly connected with the window coordinate frame. After performing several learning experiments in which we were unable to produce any generalizable knowledge, we decided to perform some experiments in transforming the axes to discover whether it was possible to simplify the results. However, the original choice of the coordinate

system was connected with the experimental environment, and the data were expressed in the coordinate system which was common to the robot and the vision system.

We performed an experiment to test different coordinate frame representations of the data: a window-oriented coordinate frame (X and Y lie along the axes of the play_window), an object-oriented coordinate frame (the vision coordinate system—Y lies along the principal axis of the object with 0,0 offsets at X_obj_start, Y_obj_start) and an actor-oriented coordinate frame (robot coordinate system—Y lies along the line of the action vector with 0,0 offsets at X_rob_start, Y_rob_start). While the original data lay in a window-oriented coordinate frame, two new sets of geometrically transformed data were constructed for the object- and the actor-oriented coordinate frame. Each of the three data-sets were then passed to Assistant:86. Both complete and pruned trees were generated for each of the three experimental conditions. Figure 6 shows the three resulting trees. The trees were analysed by looking at the number of nodes, leaves, and attribute types present in the induced tree for each experimental condition. The results are shown in Table 1.

Table 1. Table of results for coordinate frame experiment.

Parameter	Coordinate frame type		
	Window	Object	Actor
No. of nodes (pruned)	15	13	9
No. of leaves (pruned)	8	7	5
No. of null leaves	1	1	0
No. of different attributes appearing in the tree	6	4	3

Given that there is no reason for choosing one coordinate frame in preference to another, we decided to follow the principle offered by Occam's Razor which suggests that, given multiple descriptions, the simplest should be chosen, that is, low complexity is synonymous with high credibility [9]. We chose the actor-oriented coordinate frame. The presence of null leaves in the trees for both the window- and object-centred descriptions supports the choice.

4. GENERATING A MODEL

Having now ascertained an appropriate coordinate frame for describing the experiment, we next turned our attention to trying to discover any regularities in the experimental data and to discover any accompanying representation. All further experimentation uses the geometrically transformed raw data.

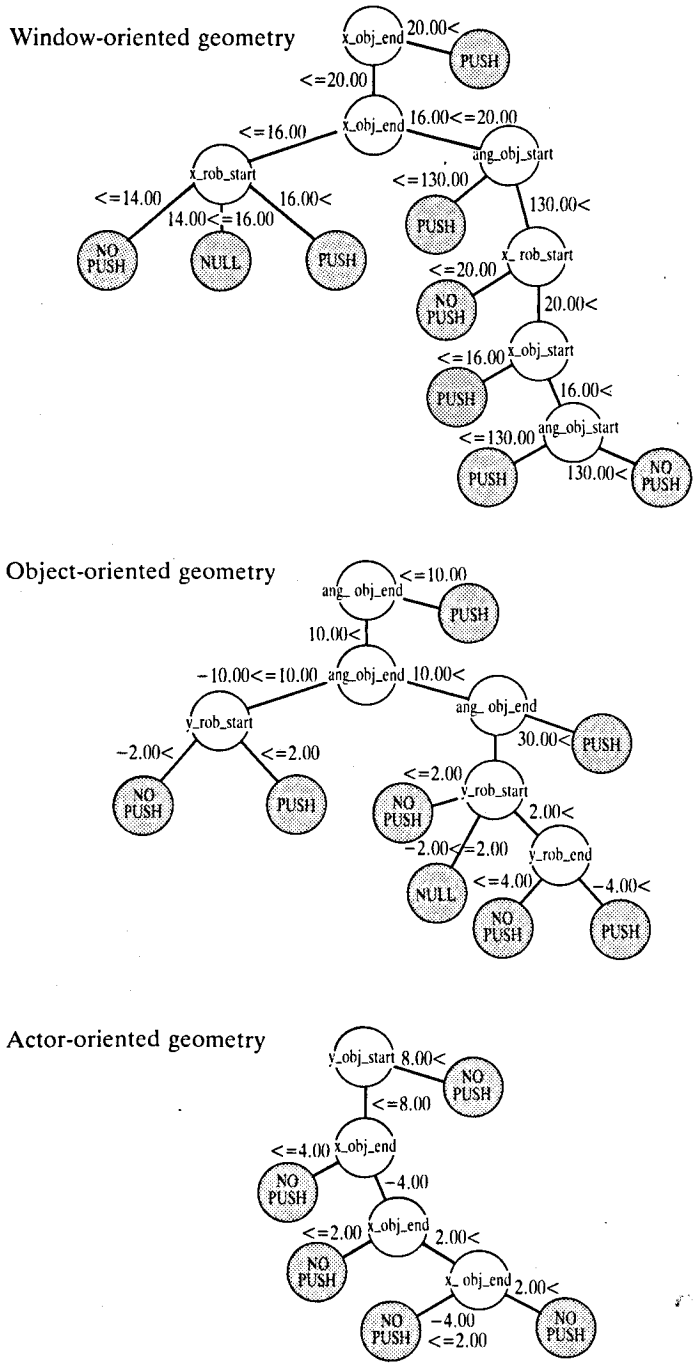


Figure 6. The decision trees shown above are for the window-oriented coordinate frame, the object-oriented coordinate frame, and the actor-oriented coordinate frame.

The learning procedure used for the development of the model consists of several modules in a loop as shown in Figure 7. Development required several iterations.

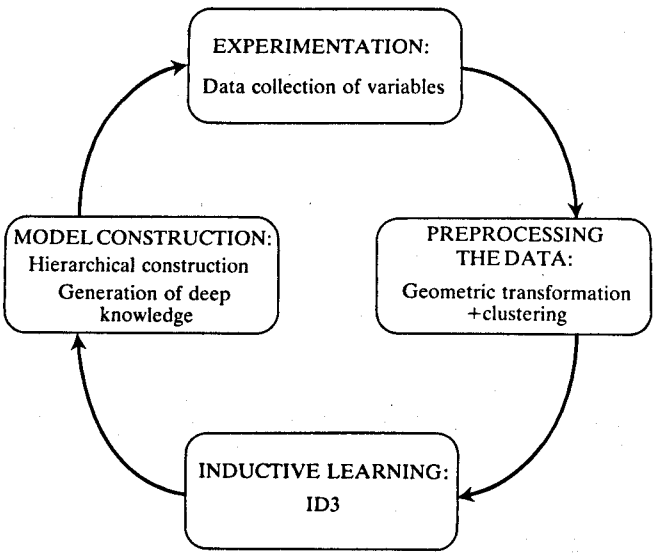


Figure 7. Summary of the sequence of operations necessary to develop the model.

Using a clustering algorithm we were able to define symbolic (logical) classes. In each iteration the machine-derived clusters were used as logical classes, each of which was named by a human oracle. The learning algorithm, Assistant:86, was used to extract the knowledge from the data in the form of rules (branches in the decision tree), which made explicit the regularities in the data (relationships between classes and attributes).

Induction was performed separately for three pairs of classes defined as changes in the *X* direction, the *Y* direction, and the orientation of the object. The three separate decision trees for changes in *Y*, *X*, and Angle of the object (*DX*, *DY*, *DA*) all shared a common root node in which *X*_obj_start = a value which corresponded to the maximum projected radius of the object (radius with which the action vector intersects). For each decision tree, this attribute value split the examples into those showing *no_change* on one branch and *change* on the other. By applying the logical operators **and** or **or**, the classes for *DX*, *DY*, and *DA* were merged into a new common class which we named PUSH (change in all directions) and NO_PUSH (no change in any direction). To test these operations, an additional experiment was performed using the PUSH and NO_PUSH classes. The result from the induction is shown as level 1 of the model depicted in Figure 8.

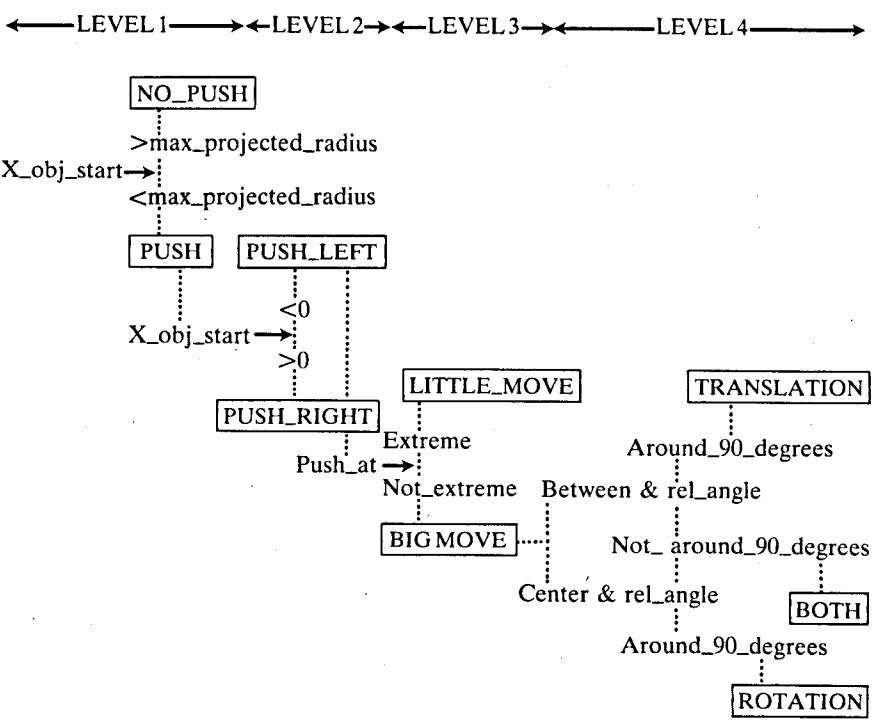


Figure 8. The final machine-learned, hierarchical, qualitative model, PANIC.

Using level 1 of the model we can predict that whenever the action vector intersects with the object then a change in position or orientation is likely to occur, i.e. it is pushed. If an intersection is unlikely then the object is left in the same place, i.e. it is not pushed.

The knowledge from level 1 was implemented as a module and used by the program that supervised the experiment. This knowledge was tested, found to be robust, and was used to collect additional data for all those situations involving object pushing. This meant that before the robot carried out an experiment, it first checked to see whether a PUSH would occur. If the model predicted that NO_PUSH would be the result, the robot calculated a new random action vector and repeated the exercise. Using level 1 to direct the robot into performing PUSH experiments, we verified that the model successfully predicted changes in position or orientation of the object and 157 new PUSH experiments were carried out. These data were used as a basis for all later learning experiments.

By using a clustering algorithm, new subclusters were found in the collected data derived from successful PUSH experiments. The data for *DX* and *DA* were grouped into two large clusters: one containing positive values and the other containing negative values. The values of *DY* were

all positive due to the choice of representation. Again, the decision trees could be merged and were used to express level 2 of the model shown in Figure 8. Level 2 of the model predicts that if an actor pushes an object on the left then it tends to move away and to the right and also to spin clockwise. If, however, the object is pushed on the right then the object moves away and to the left and tends to spin clockwise. By finding further subclusters in the data and repeating the learning procedure, two further levels were discovered which are again shown in Figure 8. Level 3 allows us to predict that if the object is pushed on the extreme left or the extreme right then we may expect a small change in rotation and translation. Alternatively, if we push near the centre of the object then we would expect a large change to occur—translational or rotational. Level 4 allows us to refine our predictions from level 3. The form of the refinement is that if we take into account the relative angle between the actor and the object, we can predict pure translation whenever we push in the centre or orthgonally. Alternatively, when we push orthogonally to one side of the object we may expect maximum rotation. Any other combination does not allow us to make a reliable prediction.

Figure 9 shows a graphical representation of the model. The position on the block along with the qualitative class name resulting from an action is indicated for all levels of the model.

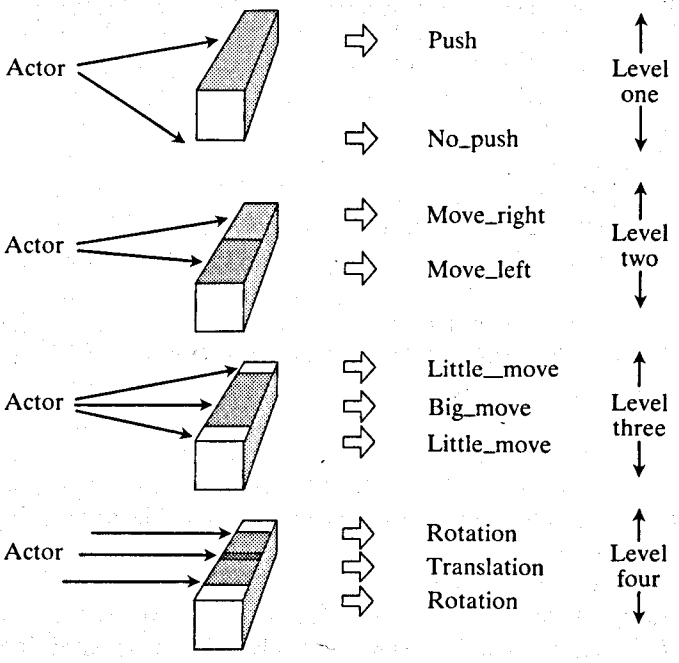


Figure 9. Graphical representation of PANIC.

5. LEVELS OF ABSTRACTION

At different levels of the hierarchy, the knowledge may be described in different ways. These descriptions may be summarized as in Figure 10.

The complete hierarchical qualitative model was named PANIC which stands for Perception and Action as NaIve Causality.

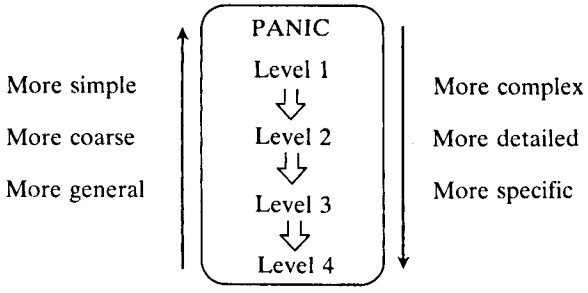


Figure 10. Knowledge of hierarchy levels of PANIC.

6. DISCUSSION AND CONCLUSION

The experiment has introduced a novel paradigm into robot learning research. The paradigm is that of causality learning in which a robot with sensors is allowed to carry out experiments in a partially random, partially directed fashion. Coupling the experimental data with a primitive machine learning system has allowed broad clusters and subclusters to be discovered in the data. When these clusters are described with symbolic names, they can be organized in the form of a hierarchical, qualitative model.

One of the important findings of the experiment was that by changing the coordinate frame it was possible considerably to simplify the model. Of the different coordinate frames tested, the actor-oriented representation proved to be the most compact. Further, it makes sense—an actor on stage describes 'stage left' relative to the actor rather than to the audience. Also, humans often talk about 'left-hand' or 'right-hand' side in terms of their own coordinate frame. To have described the coordinate frame as object-centred is logical in that an action is only an action relative to what you understand as having happened in your world, i.e. you hit the ball, you open the door, or you push the table. Given this natural tendency to describe coordinate frames for events relative to ourselves, it is not surprising that ancient man typically believed Earth to be at the heart of the universe.

There were some difficulties because the machine-learning algorithm was far from ideal for this task. This meant that a large number of iterations (machine-learning experiments) had to be peformed. The reason

was that the requirement of discovering broad classes of data which could be described in qualitative terms required a numerical, hierarchical, clustering algorithm. Because such an algorithm was not available, many of the problems in model construction related to the tools rather than to any intrinsic problem with the experimental method. Further, while the steps described here for model construction are potentially open to complete automation, some of the later stages are currently 'hand-held'. Closed-loop automation is an important goal for later work.

A second problem with the results was that some parts of the model, particularly in level 4, were supported by few examples. Because the exploration of the domain was random, the learning was therefore limited. To make the learning more effective the robot requires an explicit strategy by which it can direct its exploration towards 'interesting' clusters of results. The term 'interesting' suggests that the robot has some form of motivation. We provided such direction after discovering level 1 of the model. Although NO_PUSH was a useful class to discover (it is after all the basis for developing collision avoidance), when level 1 was implemented, it was used to direct the robot into PUSH experiments. Thus, we gave direction to the robot by knowing that it was not going to discover very much about interacting with objects if it never touched them. This simple piece of 'common sense' knowledge is, however, easy to build into the robot. The mechanism that would allow this to happen would be that the robot direct its behaviour in such a way as to increase the likely amount of sensory change. Such an idea appears commonplace in biological systems where things that are bright, loud, or quickly moving are typically more interesting than things that are not. Indeed, biological sensors appear to be specifically designed to respond more strongly to change than to steady-state stimulation.

Unfortunately, if strong sensory change was the only parameter to effect primitive motivation, the robot would work its way into a single local minimum and get stuck there. To avoid such situations, we propose that a complementary process be used which can be referred to as habituation or boredom. This would mean that whenever the robot carried out a certain number of experiments within one portion of the problem space and had found some reasonably reliable transformation there, it would get bored and would jump out of the local minimum via the random number generator. Thus each cluster of the model would have an associated weighting factor whose value would be determined by the two opposing forces of strong sensory change and habituation. Whenever the weighting factor falls below a certain level, the robot redirects itself to play with some other problem.

So far, the model is untested. It was generated using all the available data, hence no generalization has been demonstrated. Although future work must include such studies, it is worth pointing out that the model,

as with the coordinate frame makes good sense. While level 1 is almost too obvious to mention (you can push an object only if the action that you carry out intersects with some part of the object), level 2 could allow you to reason about some quite interesting problems. For example, if you want to push a car forward at which end should you stand?, or if you want to make a billiard ball spin anticlockwise on which side should you hit it? Level 2 of PANIC shows that during any push an increase in Y is very likely and that you must therefore stand at the rear of the car so that your action vector pushes the car forward. The anticlockwise spin on the billiard ball would be achieved if it were pushed on the right. Level 3 would tell you that if you wanted to hit a ball as far as possible then you should hit it in the middle, while level 4 could help in knowing how to produce pure translation or rotation movements in objects. Thus, if you wanted to push a boat out into a river in a straight line PANIC would predict that you could stand behind it and push roughly in the middle so that the relative angle between the surface of the boat and the action vector is around 90 degrees.

While the experiments described here represent some early steps taken in getting a robot to learn for itself about how to act in the world, one short-term goal will be to use PANIC-style rules to supplement a conventional robot control system. Figure 11 shows one simple example of how rules derived from level 2 of the model could be used to push an object into a desired configuration by a process of minimization. Providing the control loop steps are sufficiently small then *DY*, *DX* and Dang will all be effectively zero on completion of the control loop.

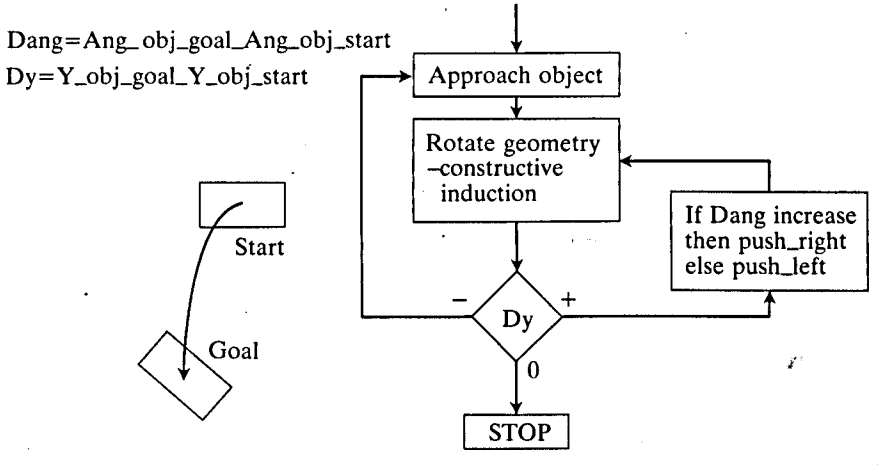


Figure 11. Use of PANIC rules for dynamic control.

Acknowledgements

The authors thank the SERC and The British Council for financial support and to the many researchers at both Ljubljiana and Glasgow who have contributed with ideas. Thanks in particular to Barry Shepherd, Ivan Bratko, and Peter Clarke.

REFERENCES

1. Cestnik, B., Kononenko, I. and Bratko, I. (1987). Assistant: 86: a knowledge-elicitation tool for sophisticated users. In *Progress in machine learning: proceedings of the EWSL 87: European working session on learning* (eds I. Bratko and N. Lavrac) pp. 31–45, Sigma Press, Wilmslow.
2. Dechter, R. and Michie, D. (1984). *Induction of plans. TIRM 84-006*, The Turing Institute, Glasgow.
3. Doran, J. E. and Michie, D. (1966). Experiments with the Graph Traverser program. *Proc. R. Soc.*, **294**, 235–59.
4. Dufay, B. and Latombe, J. C. (1983). An approach to automatic robot programming based on inductive learning. In *First Int. Symp. on Robotics Res.*, Bretton Woods.
5. Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, pp. 189–208.
6. Michie, D. and Chambers, R. A. (1968). BOXES: an experiment in adaptive control. In *Machine Intelligence 2* (eds E. Dale and D. Michie) pp. 137–52, Oliver and Boyd, Edinburgh.
7. Mowforth, P. H. and Bratko, I. (1987). AI and robotics: flexibility and integration. *Robotica*, **5**, 93–8.
8. Newell, A., Shaw, J. C. and Simon, H. A. (1960). A variety of intelligent learning in a general problem solver. In *Self-organising systems* (eds Yovits Marshall C and Scott Cameron) pp. 153–89. Pergamon, London.
9. Pearl, J. (1978). On the connection between the complexity and credibility of inferred models. *Int. J. General Systems*, **4**, pp. 255–64.
10. Quinlan, J. R. (1979). Discovering rules from large collection of examples: a case study. In *Expert Systems in the Micro Electronic Age*, (ed. D. Michie). pp. 168–201, Edinburgh University Press, Edinburgh.
11. Tate, B. A. (1976). *Project planning using a hierarchical, non-linear planner.* Artificial Intelligence Research Report 25, Edinburgh University, Edinburgh.
12. Zrimec, T. (1986). Application of logic programming to task-level programming of robots. In *AFCET proceedings: Robot and artificial intelligence*, Toulouse.

16

A Qualitative Way of Solving the Pole Balancing Problem

A. Makarovic
University of Twente,
The Netherlands

Abstract

A method is presented for designing a control rule for the well-known pole-cart system. Conventional techniques are used to model the pole-cart system by a set of differential equations. The high complexity of the model prohibits a direct derivation of the control rules. The model can be simplified by restructuring and approximating the equations. We allow *very* rough approximations, as long as the essence of the problem stays the same. Ultimately we get a very simple model. Designing a control rule for this model is an almost trivial problem.

This experiment underlines the theme that for solving problems, often only a small part of the available information is really needed. By reformulating the problem and omitting the information that is not really needed, we can reduce the complexity of the problem, with disproportionate influence on the search space of potential solutions.

1. HISTORY

The pole balancing problem was used by Michie and Chambers (1968) as a test case in the area of machine learning. Since then it has become a widely accepted benchmark problem in this area of research.

During a working visit at the Jozef Stefan Institute in Ljubljana, Yugoslavia, the author was introduced to the pole balancing problem as a means of investigating a qualitative simulator called QSIM (Kuipers 1986). After recognizing QSIM as not being an appropriate tool for dealing with the pole balancing problem, the author successfully developed his own approach. At that time a preliminary report was written (Makarovic 1987). A more detailed description will be given here of the way that the pole balancing problem was solved.

2. DESCRIPTION OF THE PROBLEM

A graphical illustration of the pole-cart system is given in Figure 1. A pole is mounted on top of the cart. Optionally a second pole can be

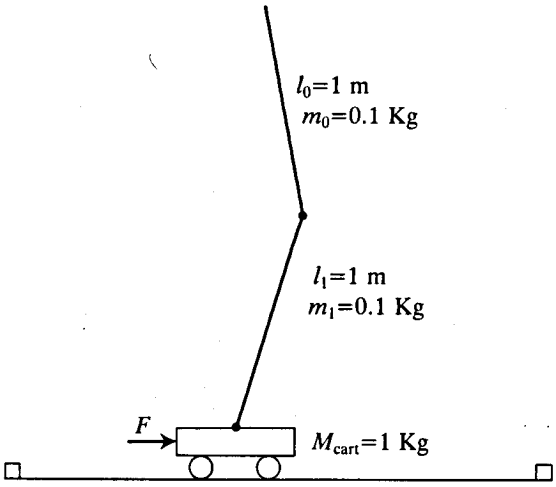


Figure 1. The pole-cart system.

mounted on top of the first pole. All movements are in one plane, so the problem is stated in a two-dimensional world. The problem is to balance the poles by selecting the appropriate direction of the applied force. The magnitude of the force is fixed and never zero. There are only two possible actions one can select from: accelerating forward, or accelerating backward. This type of control is often called 'bang-bang control', because of its abrupt nature.

In order to be able to choose an appropriate action, the state of the system must be known. Therefore the position and velocity of the cart, and the angles and angular velocities of the poles are measured periodically. In fact such measurements only compare the value to be measured with some reference values. All that is known after the measurement is in which interval the value must be. Each variable has a private set of reference values dividing its quantity space into typically only a few intervals. Each time a measurement is performed, an appropriate action has to be selected.

The pole balancing problem consists of

- how to divide the quantity space of each variable into a small set of intervals;
- what action to select for each combination of intervals describing the state of the pole-cart system

in a way such that

- the poles are balanced, i.e. do not fall;
- the cart does not leave a predetermined limited area.

See Michie and Chambers (1968), Makarovic (1987) and Sammut (1988) for more information about the pole balancing problem.

3. MODELLING THE POLE-CART SYSTEM

By applying physical laws to the pole-cart system we can derive a set of equations constraining the possible behaviours of that system. A problem that can occur is that the initial set of constraints is not complete. Completeness can be verified by comparing the number of independent equations with the number of unknown variables. These numbers must be equal because we know that the pole-cart system must behave deterministically. Missing equations can be found by applying the same technique on parts of the system.

The meaning of variables is as follows.

Relevant properties of the pole-cart system:

F: force acting [N]
G: gravity constant = 9.8 [m/s^2]
θ: angle of a pole relative to the vertical, anticlockwise positive [rad]
P: position [m]
M: mass [kg]
L: length [m]
Delay: time needed to reach the desired reference value [s]

Designator indicating part and direction of a property:

t: top of the pole
m: middle of the pole
b: bottom of the pole
x: horizontal component, positive to the right
y: vertical component, positive upward

Designator indicating the object:

0: the upper pole
1: the lower pole
c: the cart

Designator indicating a sub-property:

ref: the (positive) boundary dividing the quantity space
osc: the amplitude of oscillation

So for example 'F_{by0}' means the upward component of the force acting on the bottom part of the upper pole.

A set of physics laws constraining the behaviour of the pole-cart system is set up. The forces acting:

$$
\begin{aligned}
F_{tx0} &= 0 && \text{no external force acting on the top}\\
F_{ty0} &= 0 &&\\
F_{mx0} &= 0 && \text{no horizontal component of gravity}\\
F_{my0} &= -M_0 \cdot G && \text{gravity}\\
F_{bx0} &&&\\
F_{by0} &&&\\
F_{tx1} &= -F_{bx0} && \text{action = -reaction}\\
F_{ty1} &= -F_{by0} &&\\
F_{mx1} &= 0 &&\\
F_{my1} &= -M_1 \cdot G &&\\
F_{bx1} &&&\\
F_{by1} &&&\\
F_{xc} &= F_{x,applied} - F_{bx1} &&\\
F_{yc} &= -F_{by1} + F_{supporting} = 0 && (1)
\end{aligned}
$$

According to Newton, acceleration equals the sum of the forces (torques) acting, divided by (rotational) mass. So for translations:

$$
\begin{aligned}
\ddot{P}_{mx0} &= \frac{F_{tx0} + F_{mx0} + F_{bx0}}{M_0}\\
\ddot{P}_{my0} &= \frac{F_{ty0} + F_{my0} + F_{by0}}{M_0}\\
\ddot{P}_{mx1} &= \frac{F_{tx1} + F_{mx1} + F_{bx1}}{M_1} && (2)\\
\ddot{P}_{my1} &= \frac{F_{ty1} + F_{my1} + F_{by1}}{M_1}\\
\ddot{P}_{xc} &= \frac{F_{xc}}{M_c}\\
\ddot{P}_{yc} &= \frac{F_{yc}}{M_c}
\end{aligned}
$$

and for rotations:

$$\ddot{\theta}_0 = \frac{1/2 \cdot L_0 \cdot (+F_{bx0} \cos\theta_0 + F_{by0} \sin\theta_0 - F_{tx0} \cos\theta_0 - F_{ty0} \sin\theta_0)}{1/12 \cdot L_0^2 \cdot M_0} \quad (3)$$

$$\ddot{\theta}_1 = \frac{1/2 \cdot L_1 \cdot (+F_{bx1} \cos\theta_1 + F_{by1} \sin\theta_1 - F_{tx1} \cos\theta_1 - F_{ty1} \sin\theta_1)}{1/12 \cdot L_1^2 \cdot M_1} \quad (4)$$

The poles are interconnected so the position of the top part of the lower pole must be equal to the position of the bottom part of the upper

pole. Because the positions are *always* equal, the velocities and accelerations must be equal as well.

$$\begin{array}{lll} P_{bx0} = P_{tx1} & \dot{P}_{bx0} = \dot{P}_{tx1} & \ddot{P}_{bx0} = \ddot{P}_{tx1} \\ P_{by0} = P_{ty1} & \dot{P}_{by0} = \dot{P}_{ty1} & \ddot{P}_{by0} = \ddot{P}_{ty1} \\ P_{bx1} = P_{mxc} & \dot{P}_{bx1} = \dot{P}_{mxc} & \ddot{P}_{bx1} = \ddot{P}_{mxc} \\ P_{by1} = P_{myc} & \dot{P}_{by1} = \dot{P}_{myc} & \ddot{P}_{by1} = \ddot{P}_{myc} \end{array} \tag{5}$$

A pole is a rigid body, so there exists a fixed relation between the positions of the top, middle, and bottom parts of a pole, and the angle of that pole. Similar constraints can (and must) be derived for velocities and accelerations of these parts.

$$\begin{array}{l} P_{tx0} = P_{mx0} - 1/2 \cdot L_0 \cdot \sin\theta_0 \\ P_{ty0} = P_{my0} + 1/2 \cdot L_0 \cdot (1 - \cos\theta_0) \\ P_{bx0} = P_{mx0} + 1/2 \cdot L_0 \cdot \sin\theta_0 \\ P_{by0} = P_{my0} - 1/2 \cdot L_0 \cdot (1 - \cos\theta_0) \\ P_{tx1} = P_{mx1} - 1/2 \cdot L_1 \cdot \sin\theta_1 \\ P_{ty1} = P_{my1} + 1/2 \cdot L_1 \cdot (1 - \cos\theta_1) \\ P_{bx1} = P_{mx1} + 1/2 \cdot L_1 \cdot \sin\theta_1 \\ P_{by1} = P_{my1} - 1/2 \cdot L_1 \cdot (1 - \cos\theta_1) \end{array} \tag{6}$$

4. VERIFICATION OF THE SOLUTION TO BE FOUND

It may seem strange to care about verification of a solution before the solution itself is found, but this order makes finding a solution easier. If we agree on a verification we can try to find a solution that satisfies the verification and forget about the original problem. However, the verification must be very reliable for this purpose.

The most convincing verification would be testing the control rule in practice with a real pole-cart system. For practical reasons we agreed on empirical verification with a numerical simulator of the pole-cart system.

In order to construct a numerical simulator we had to rewrite the initial constraints as explicit differential equations, i.e. the highest derivatives equal explicit functions of other variables, as shown below.

After tedious but straightforward manipulation, in which non-relevant local variables were eliminated, we derived for the upper pole:

$$\ddot{\theta}_0 = SemiConst_0 \cdot \left(\begin{array}{l} + \ddot{P}_{xc} \cos\theta_0 + G \sin\theta_0 + \\ -\dfrac{3M_1 + 6M_0}{2M_1 + 6M_0} \cos(\theta_1 - \theta_0)(\ddot{P}_{xc} \cos\theta_1 + G \sin\theta_1) + \\ -\dfrac{3M_0}{2M_1 + 6M_0} \tfrac{1}{2} L_0 \sin(2(\theta_0 - \theta_1))\, \dot{\theta}_0^2 + \\ - L_1 \sin(\theta_0 - \theta_1)\, \dot{\theta}_1^2 \end{array} \right) \tag{7}$$

with:

$$SemiConst_0 = \frac{6M_1 + 18M_0}{(4M_1 + (3 + 9\sin^2(\theta_0 - \theta_1))M_0)L_0} \tag{8}$$

for the lower pole:

$$\ddot{\theta}_1 = SemiConst_1 \cdot \begin{pmatrix} +(M_1 + 2M_0)(\ddot{P}_{xc}\cos\theta_1 + G\sin\theta_1) + \\ -\tfrac{3}{2}M_0\cos(\theta_0 - \theta_1)(\ddot{P}_{xc}\cos\theta_0 + G\sin\theta_0) + \\ +\tfrac{3}{4}M_0L_1\sin(2(\theta_0 - \theta_1))\dot{\theta}_1^2 + \\ +M_0L_0\sin(\theta_0 - \theta_1)\dot{\theta}_0^2 \end{pmatrix} \tag{9}$$

with:

$$SemiConst_1 = \frac{6}{(4M_1 + (3 + 9\sin^2(\theta_0 - \theta_1))M_0)L_1} \tag{10}$$

and for the cart:

$$\ddot{P}_{xc} = \frac{F_{x,applied}}{M_c} - \ldots \tag{11}$$

5. REDUCING COMPLEXITY AND ADDING CAUSALITY

For finding a solution we prefer using the explicit differential equations instead of the initial set of constraints. Both models describe the behaviour of the pole-cart system equally well, but there are some other differences. Unlike the initial set of constraints, the explicit differential equations.

(1) are less complex;

(2) contain a kind of causal information implicit in their structure. This extra information must be incorporated in the *algorithm* of the numerical simulator.

Although the complexity is reduced considerably, the complexity of the model is still much too high to derive a control rule. The causal structure of the differential equations enables us to recognize which interacting influences are *comparable* because of having similar effects. Because of this extra knowledge we can distinguish between important and less important influences. In the approximations developed below influences of minor importance are neglected. Until now we have simplified the structure of the equations as much as possible, without sacrificing the accuracy of the model. The complexity, however, is still much too high to be able to derive a control rule. From now on we will

have to sacrifice some accuracy in order to reduce the complexity further.

5.1. Hypothesis about the operating region

The pole-cart system is an unstable system. The simplest case is to keep the system in equilibrium. Therefore we search for a solution where the system is controlled in such a way that it will stay in a close region around equilibrium. The system is in equilibrium when all angles, velocities, and accelerations are zero.

5.2. Neglecting θ-dependency of *SemiConst*

Approximation The expressions denoted by $SemiConst_0$ and $SemiConst_1$ are approximated by real constants $Const_0$ and $Const_1$.

Justification The dependency of *SemiConst* upon the angles ($= (dSemiConst/d\theta)$) is very small compared to the magnitude of *SemiConst.*

Effect The magnitude of the angular accelerations will be slightly smaller in reality as predicted by the approximated model. The sign is never affected.

So by now we have:

$$Const_0 = \frac{6M_1 + 18M_0}{(4M_1 + 3M_0) \cdot L_0} \tag{12}$$

$$Const_1 = \frac{6}{(4M_1 + 3M_0) \cdot L_1} \tag{13}$$

5.3. Neglecting the dependency of $\dot{\theta}$'s on $\ddot{\theta}$'s

Approximation The last two terms of both equations (7) and (9) are neglected, i.e. approximated by zero.

Justification All these terms contain a factor like $\sin(\theta_0 - \theta_1) \cdot \dot{\theta}^2$. Because both angles and angular velocities are assumed small, that whole factor is *small* · $small^2$ = *verysmall.* The rest of these terms is of the same order of magnitude as the terms not being neglected.

Effect The effect of this approximation is probably comparable with rounding errors while computing $\ddot{\theta}$.

5.4. Approximating ugly functions

Approximation Goniometric functions are approximated by the first two terms of the Taylor expansion, so $\cos(\theta) \approx 1$ and $\sin(\theta) \approx \theta$.

Justification The operating region of θ is supposed to be close to zero.

Effect The effect of this approximation is probably comparable with rounding errors while computing $\ddot{\theta}$.

Doing all these approximations we get for the upper pole:

$$\ddot{\theta}_0 = \frac{6M_1 + 18M_0}{(4M_1 + 3M_0)L_0} \cdot \begin{pmatrix} +\ddot{P}_{xc} + G\theta_0 + \\ -\dfrac{3M_1 + 6M_0}{2M_1 + 6M_0}(\ddot{P}_{xc} + G\theta_1) \end{pmatrix} \tag{14}$$

and for the lower pole:

$$\ddot{\theta}_1 = \frac{6}{(4M_1 + 3M_0)L_1} \cdot \begin{pmatrix} +(M_1 + 2M_0)(\ddot{P}_{xc} + G\theta_1) + \\ -\frac{3}{2}M_0(\ddot{P}_{xc} + G\theta_0) \end{pmatrix} \tag{15}$$

Restructuring these equations results for the upper pole in:

$$\ddot{\theta}_0 = \frac{3}{(4M_1 + 3M_0)L_0} \cdot \begin{pmatrix} -M_1 \cdot \ddot{P}_{xc} + \\ (2M_1 + 6M_0) \cdot G \cdot \theta_0 + \\ (-3M_1 - 6M_0) \cdot G \cdot \theta_1) \end{pmatrix} \tag{16}$$

and for the lower pole in:

$$\ddot{\theta}_1 = \frac{3}{(4M_1 + 3M_0)L_1} \cdot \begin{pmatrix} (2M_1 + M_0) \cdot \ddot{P}_{xc} + \\ (2M_1 + 4M_0) \cdot G \cdot \theta_1 + \\ -3M_0 \cdot G \cdot \theta_0 \end{pmatrix} \tag{17}$$

6. CHOOSING A MEANS OF CONTROL

In order to control the pole-cart system there must be a way of influencing it. The remaining influences of this system are analysed in what follows:

Based on equations (16) and (17) a graphic representation of the simplified pole-cart system is presented in Figure 2. In order to be able to compare weight factors, we filled in the parameters of the pole-cart system. We assume the poles having equal lengths (1 m) and equal masses. The mass of the cart M_c is large compared with the masses of the poles. This model is called *semi* qualitative because it is very useful for investigating the nature of the pole-cart system, but it is still too complex for one to be able to derive a control rule from it.

We can recognize in this model that:

1. The acceleration of the cart is under direct control of the applied force.
2. The acceleration of the cart directly influences the angular accelerations of both poles. The influence upon the lower pole is three times as great as upon the upper pole.

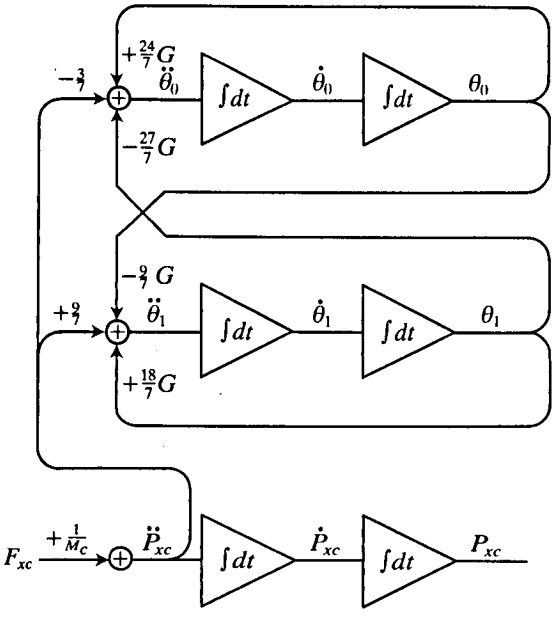


Figure 2. Semi-qualitative model of the pole-cart system.

3. The angle of the lower pole influences the acceleration of the upper pole.
4. The angle of the upper pole influences the acceleration of the lower pole.
5. There are three positive feedback loops, causing the very unstable nature of the poles.
   (a) The angle of the upper pole positively influences the acceleration of that pole.
   (b) The angle of the lower pole positively influences the acceleration of that pole.
   (c) The angle of the lower pole negatively influences the acceleration of the upper pole, and the angle of the upper pole negatively influences the acceleration of the lower pole. This is the reason why the poles tend to fall in opposite directions.

So through the applied force we can control the acceleration of the cart, but how should we control the angles of the poles? There are two possibilities:

1. By controlling the acceleration of the cart we can control the lower pole, and by controlling the lower pole we can control the upper pole.

$$\ddot{P}_{xc} \xrightarrow{+\frac{9}{7}} \ddot{\theta}_1 \xrightarrow{\int dt} \dot{\theta}_1 \xrightarrow{\int dt} \theta_1 \xrightarrow{-\frac{27}{7}G} \ddot{\theta}_0 \xrightarrow{\int dt} \dot{\theta}_0 \xrightarrow{\int dt} \theta_0$$

2. By controlling the acceleration of the cart we can control the upper pole, and by controlling the upper pole we can control the lower pole.

$$\ddot{P}_{xc} \xrightarrow{-\frac{3}{7}} \dot{\theta}_0 \xrightarrow{\int dt} \ddot{\theta}_0 \xrightarrow{\int dt} \theta_0 \xrightarrow{-\frac{9}{7}G} \ddot{\theta}_1 \xrightarrow{\int dt} \dot{\theta}_1 \xrightarrow{\int dt} \theta_1$$

Obviously the first alternative has a stronger means of control than the second. Therefore we have selected the means of control indicated in Figure 3.

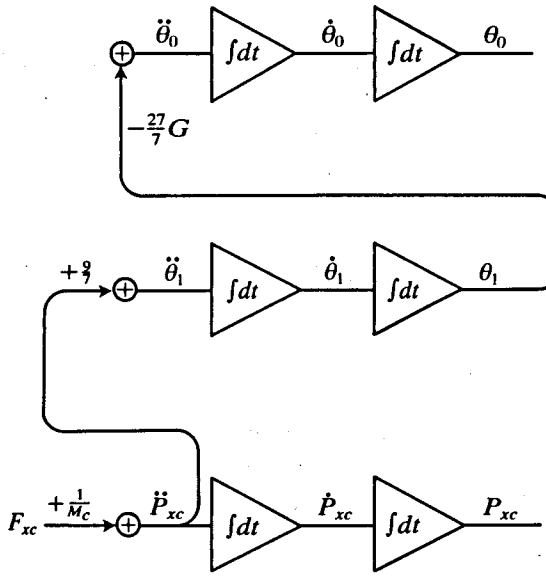


Figure 3. Dominant path of control.

To summarize, the applied force, that is determined by the control rule directly, causes the cart to accelerate. This acceleration (integrated over time) influences the velocity of the cart, and that velocity influences the position of the cart. The acceleration of the cart also influences the angular accelerations of both poles. The angles of both poles influence the angular accelerations of each other.

As a means of control we select the strongest path of influence, i.e. the lower pole controls the upper pole. By choosing a suitable operating region we must assure that this path of influence is indeed dominant.

7. CONTROLLING THE CART WHILE BALANCING POLES

The next problem to be solved is that we have to control both the movements of the cart and the movements of the poles, by using only one control variable. So the problem seems to be over-determined and thus

not solvable. However, a closer investigation of the poles reveals that there is not only one unique equilibrium where all poles are oriented vertically. When the cart is accelerating appropriately, both poles (equally) askew can be in equilibrium as well. The solution to our problem is that we do not balance the poles exactly around the vertical, but slightly askew. On average the cart will accelerate to the right if the poles are leaning to the right, and accelerate to the left if the poles are leaning to the left. Note that this behaviour will only occur under the assumptions made, i.e. the poles must be *successfully balanced* all the time, as explained below.

By controlling the acceleration of the cart $\ddot{P}_{xc}$ both the angles of the poles, and the position of the cart are influenced. While balancing the poles, how do we prevent the cart from drifting away?

Therefore we have to investigate equations (16) and (17) again. The poles being steady around the vertical is not a unique equilibrium. When the poles are in equilibrium, by definition their angular accelerations must be zero.

$$0 = \ddot{\theta}_0 = \begin{pmatrix} -\frac{3}{7} \cdot \ddot{P}_{xc} + \\ +\frac{24}{7} \cdot G \cdot \theta_0 + \\ -\frac{27}{7} \cdot G \cdot \theta_1 \end{pmatrix} \tag{18}$$

$$0 = \ddot{\theta}_1 = \begin{pmatrix} +\frac{9}{7} \cdot \ddot{P}_{xc} + \\ -\frac{9}{7} \cdot G \cdot \theta_0 + \\ +\frac{18}{7} \cdot G \cdot \theta_1 \end{pmatrix} \tag{19}$$

So if the poles are in equilibrium we know that:

$$\theta_0 = \theta_1 \approx -\frac{\ddot{P}_{xc}}{G} \tag{20}$$

(The exact solution would be $\theta_0 = \theta_1 = \tan(-(\ddot{P}_{xc}/G))$. This similarity gives us some confidence that we have not made errors in the previous derivations.) The solution to our control problem is that we do not balance the successfully balanced poles exactly around the vertical, but slightly askew, as stated earlier (see Figure 4).

8. CONTROLLING THE POLE-CART SYSTEM

The qualitative model we have by now is very simple. The applied force influences the angular acceleration of the lower pole. The angle of the lower pole influences the angular acceleration of the upper pole. The angle of the upper pole influences the average acceleration of the cart. So

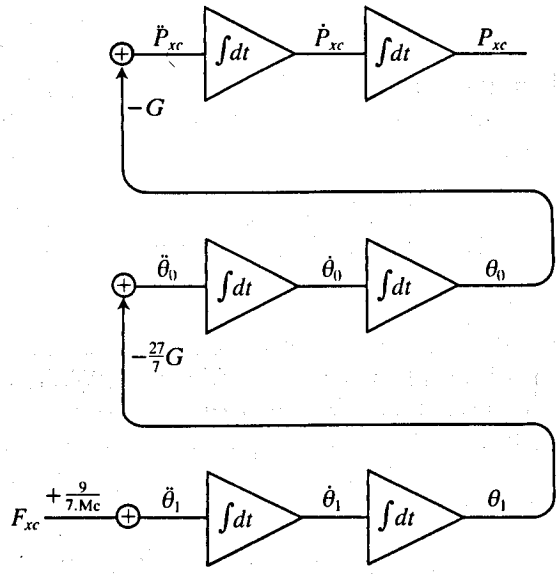


Figure 4. Modelling drift behaviour assuming successfully balanced poles.

we have one chain containing six integrators interconnected by weight factors with different signs.

$$F_{xc} \xrightarrow{+C_1} \ddot{\theta}_1 \xrightarrow{\int dt} \dot{\theta}_1 \xrightarrow{\int dt} \theta_1 \xrightarrow{-C_2} \ddot{\theta}_0 \xrightarrow{\int dt} \dot{\theta}_0 \xrightarrow{\int dt} \theta_0 \xrightarrow{-C_3} \ddot{P}_{xc} \xrightarrow{\int dt} \dot{P}_{xc} \xrightarrow{\int dt} P_{xc}$$

Controlling one integrator can be done as follows: if the value at the output is too high, decrease that value by putting a negative value at the input of the integrator. If the value is too low, increase it with a positive value at the input. Controlling a whole chain of integrators can be done by recursively applying the above rule.

If the value of the position of the cart is too low we must increase that value by making the velocity equal to a positive reference value. Because of delay in control the cart will not stop immediately when reaching the middle. In order to limit the overshoot (amplitude of oscillations) we must limit the velocity as well. So even if the position is still too low but the velocity has reached its reference value, we should not accelerate the cart any more. This theory is not restricted to position and velocity, but can be applied to other variables as well.

In order to prevent the cart from going outside its operating region, we think it is safest to keep its position somewhere in the middle. We consider this as our goal.

By recursively computing a sub-goal from the difference between the current situation and the current goal, we can derive the ultimate sub-goal. The ultimate sub-goal determines the sign of the force to be applied on the cart. This sign is under direct control.

Note that (only) the *signs* of the weight factors interconnecting the integrator sections are of crucial importance for deriving the correct control rule, as will appear in the more detailed recapitulation which follows.

8.1. Controlling one integrator

Controlling one integrator is easy. As long as the output value is too high, decrease that value by putting a negative value on the input of that integrator. If the output value is too low, then increase that value by putting a positive value on the input.

8.2. Controlling a chain of integrators

By controlling the input of the first integrator, with some delay we can control the output of that integrator. That output multiplied by a weight factor is connected to the next integrator, so we get the same problem back for a string that is one (integrator) element shorter. Recursively we can propagate this means of control through the whole chain.

8.3. Dividing the quantity space

To be able to control an integrator we must know if the value at the output is too high or too low, relative to a desired reference value. Therefore we have to divide the quantity space of that output into two intervals, the reference value being the boundary.

Now we will have to distinguish between integrators having a successor and integrators having none. Integrators having a successor are used for controlling their successor, and will therefore need both a positive and a negative reference value. The integrator at the end only has to stay within a predetermined region, the middle being the most safe. So one reference value must be sufficient for that integrator.

So the quantity space of P_{xc} is divided into two intervals called 'positive' and 'negative', zero being the boundary. The quantity spaces of all the other variables are divided into three intervals, called 'big positive', 'small' and 'big negative'. Each variable has two private boundaries: $+ Var_{ref}$ and $- Var_{ref}$. Because the problem is symmetrical around 0 the solution should be symmetrical as well.

8.4. The control rule

As stated earlier, our qualitative model is:

$$F_{xc} \xrightarrow{+c_1} \ddot{\theta}_1 \xrightarrow{\int dt} \dot{\theta}_1 \xrightarrow{\int dt} \theta_1 \xrightarrow{-c_2} \ddot{\theta}_0 \xrightarrow{\int dt} \dot{\theta}_0 \xrightarrow{\int dt} \theta_0 \xrightarrow{-c_3} \ddot{P}_{xc} \xrightarrow{\int dt} \dot{P}_{xc} \xrightarrow{\int dt} P_{xc}$$

Our goal is to keep the position of the cart somewhere in the middle, thereby minimizing the risk of getting outside the predetermined region.

$$
\begin{array}{ll}
P_{xc,ref} = 0 & \textit{goal} \\
\dot{P}_{xc,ref} = |\dot{P}_{xc,ref}| \cdot sign(P_{xc,ref} - P_{xc}) & \textit{integrator control rule} \\
\ddot{P}_{xc,ref} = sign(\dot{P}_{xc,ref} - \dot{P}_{xc}) & \textit{integrator control rule} \\
\theta_{0,ref} = |\theta_{0,ref}| \cdot -sign(\ddot{P}_{xc,ref}) & \textit{negative proportionality} \\
\dot{\theta}_{0,ref} = |\dot{\theta}_{0,ref}| \cdot sign(\theta_{0,ref} - \theta_0) & \textit{integrator control rule} \\
\ddot{\theta}_{0,ref} = sign(\dot{\theta}_{0,ref} - \dot{\theta}_0) & \textit{integrator control rule} \\
\theta_{1,ref} = |\theta_{1,ref}| \cdot -sign(\ddot{\theta}_{0,ref}) & \textit{negative proportionality} \\
\dot{\theta}_{1,ref} = |\dot{\theta}_{1,ref}| \cdot sign(\theta_{1,ref} - \theta_1) & \textit{integrator control rule} \\
\ddot{\theta}_{1,ref} = sign(\dot{\theta}_{1,ref} - \dot{\theta}_1) & \textit{integrator control rule} \\
F_{xc,applied} = |F_{xc,applied}| \cdot sign(\ddot{\theta}_{1,ref}) & \textit{positive proportionality}
\end{array}
\qquad (21)
$$

Note that this control algorithm is very suitable for explanation. For each subgoal you can tell *why* you try to achieve it, and *how* you try to achieve it. This control algorithm can be written explicitly as a control rule by eliminating the sub-goals.

IF $\dot{\theta}_1$ = big positive THEN Push Left
IF $\dot{\theta}_1$ = big negative THEN Push Right
IF $\dot{\theta}_1$ = small
THEN IF θ_1 = big positive THEN Push Left
IF θ_1 = big negative THEN Push Right
IF θ_1 = small
THEN IF $\dot{\theta}_0$ = big positive THEN Push Right
IF $\dot{\theta}_0$ = big negative THEN Push Left
IF $\dot{\theta}_0$ = small
THEN IF θ_0 = big positive THEN Push Right
IF θ_0 = big negative THEN Push Left
IF θ_0 = small
THEN IF $\dot{P}_{xc}$ = big positive THEN Push Right
IF $\dot{P}_{xc}$ = big negative THEN Push Left
IF $\dot{P}_{xc}$ = small
THEN IF P_{xc} = positive THEN Push Right
IF P_{xc} = negative THEN Push Left

9. CHOOSING SYSTEM PARAMETERS

By now we know that we must divide the quantity spaces of all variables (except at the end of the string) into three regions: 'too negative', 'too small', and 'too positive'. We also know which control decision to take in each case. However, we do not know where exactly to choose the boundaries of the intervals. A theory is now presented describing the relationships between:

(1) the reference values;

(2) the sample rate;

(3) the propagation-delay of control;

(4) the amplitude of oscillations.

Using this theory may help in finding a satisfactory compromise between conflicting system properties.

In 1987, when the control rule was derived, this theory was not developed that far. For a quick empirical verification of the control rule, we used a guided 'trial and error' method for finding a suitable set of system parameters. By studying the behaviour leading to a failure we analysed the *reasons* of the failure. The intuitions behind the theory given were used to find a better set of system parameters. After only a few iterations the poles did not fall any more, and no long-term dangerous tendencies could be observed.

However, we do not consider the set of system parameters obtained this way, suitable to be used in a *good* (*reliable*) design. Objections against such a 'trial and error' method are:

1. The safety margin of the solution found may be insufficient in practice.
2. It is dangerous to extrapolate from a finite simulation run to characterizations involving behaviour of infinite duration.
3. It is impossible to test for all initial states in some operating regions. Assuming continuity and smoothness when using a sampled bang-bang controller may be dangerous as well.
4. It is unclear how much luck is involved in finding a solution. Little is known about the existence of a solution and little is known about the rate of convergence.

Based on the following theory, a better choice of system parameters can be made.

9.1. A qualitative model of a controlled integrator

The basic rule for controlling an integrator is: put the positive input reference on the input of the integrator if the output is too low, and the negative input reference if the output is too high.

If there is some delay in contol then the output of the integrator will *oscillate* (=periodically overshoot) around the desired output reference. The amplitude of oscillation approximately equals the distance that can be traversed at 'input reference' speed, during the delay time. Except for the first integrator section, this is quite a pessimistic approximation because the smooth transitions in reality are now being modelled by abrupt transitions.

$$X_{osc} \approx \dot{X}_{ref} \cdot Delay_X \tag{22}$$

When the control algorithm decides that it wants to change the sign of

the desired output reference value, this change will take some time to be realized. The extra delay time needed for this change approximately equals the distance between the output reference positions divided by the 'input reference' velocity. Stated another way, the extra delay time is proportional to the amplification of the integrator section. The delay time relative to the control module is approximated by the sum of the individual extra delay times of all the integrator sections (see Figure 5).

$$Delay_X \approx \frac{2 \cdot X_{ref}}{\dot{X}_{ref}} + Delay_{\dot{X}} \tag{23}$$

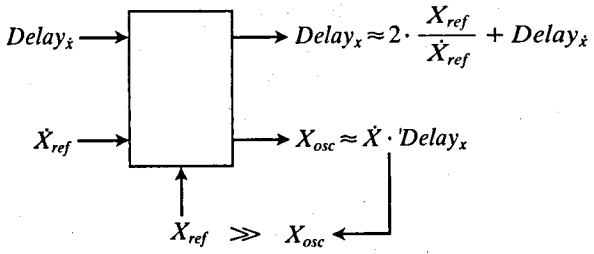


Figure 5. Modelling delay time and amplitude of oscillation (see (22) and (23) in text).

9.2. Consequences of control parameters on system properties

The successfully balanced pole-cart system can be modelled by a string of connected integrators as shown in Figure 6.

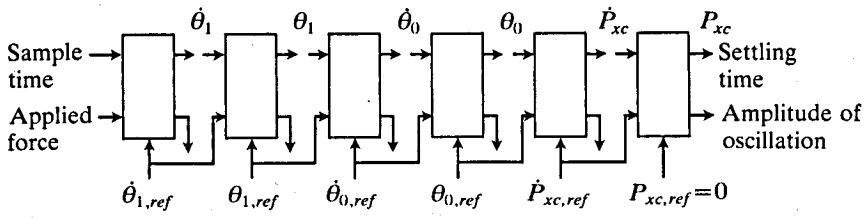


Figure 6. Delay time and oscillation in the pole-cart system.

In deriving our control rule we assumed our selected path of control to be dominant. Now we have to choose the reference values in such a way that this will indeed be the case.

$$\frac{F_{xc,ref}}{M_c} \gg 2 \cdot G \cdot \theta_{1,ref} + G \cdot \theta_{0,ref} \tag{24}$$

$$9 \cdot G \cdot \theta_{1,ref} \gg \frac{F_{xc,ref}}{M_c} + 8 \cdot G \cdot \theta_{0,ref} \tag{25}$$

If we assume that $\gg$ means at least two times greater, then we can derive that:

$$\frac{F_{xc,ref}}{M_c} : G \cdot \theta_{1,ref} : G \cdot \theta_{0,ref} \geqslant 70:18:1 \tag{26}$$

satisfying the above conditions. In this derivation we neglected the influence of oscillations, therefore we have to keep:

$$\theta_{0,osc} \ll \theta_{0,ref} \qquad \theta_{1,osc} \ll \theta_{1,ref} \tag{27}$$

The model represented by Figure 6 is a network of constraints restricting the design choices of the reference values somewhat. The remaining freedom can be used for finding a compromise of the desired system properties. This model can provide us with a good qualitative understanding of the consequences of the design choices, enabling us to see which properties we have to choose between. We would like to achieve with a low sample rate: a large operating region for the poles, a limited amplitude of oscillation at the output, the output quickly reaching its desired value, etc.

In order to keep the selected path of control dominant, the amplification of some integrator sections must be limited. ($G \cdot \theta_{0,ref} : G \cdot \theta_{1,ref} : (F_{xc,ref}/M_c) \leqslant 1:18:70$.) However, the reference values must be greater than the amplitudes of oscillation putting a lower boundary upon amplification. In order for a solution to exist, the sample rate must be sufficiently high.

Furthermore we assumed that $\sin(\theta) \approx \theta$ and $\cos(\theta) \approx 1$. Therefore $\theta_{1,ref}$ should not be chosen too large, i.e. not much greater than 0.5 [Rad] $\approx$ 30[Deg].

If the combined amplification of a string of integrators is already determined, choosing equal individual amplifications will introduce less delay than choosing totally different amplifications.

10. CONCLUSIONS

1. Our approach of problem solving by using qualitative models is successful for finding a solution of the pole balancing problem. We showed how to control a pole-cart system with two approximately equal poles stacked on each other.
2. The qualitative models created are very suitable for explanation. They provide us with a good understanding of how the pole-cart system behaves, and why our control rule should be as it is.

3. On both theoretical and empirical grounds we have some confidence that our control rule is correct if we satisfy the assumptions made.
4. The assumptions made have to be satisfied by choosing suitable system parameters like:
   (a) the magnitude of the applied force;
   (a) the reference values used for comparison in measurements;
   (c) sample rate;
   (d) the operating region;

   We have both theoretical and empirical evidence that these assumptions can be satisfied for two equal poles. If the upper pole is made shorter, relative to the lower pole, satisfying these assumptions will be more difficult and may become impossible.
5. Changing the length and mass of the poles will change the strength of the interacting influences in the pole-cart system. Shortening the upper pole too much may change the dominant path of influence, and may therefore change the choice of the selected path of control. This would result in other qualitative models, resulting in another control rule and other assumptions to be satisfied.
6. A *really good* set of system parameters still has to be found. The theory presented may be helpful for finding such a set.

REFERENCES

Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence* **29**, pp. 289–338.

Makarovic, A. (1987). *Pole balancing as a benchmark problem for qualitative modelling.* 1987 Internal Report number 4953 Jozef Stefan Institute, Ljubljana, Yugoslavia.

Michie, D. and Chambers, R. A. (1968). Boxes: an experiment in adaptive control. *Machine Intelligence* **1**, pp. 137–52 Edinburgh University Press.

Sammut, C. (1988). Experimental results from an evaluation of algorithms that learn to control dynamic systems. *Proc. Fifth Intern. Conf. Machine Learning* (ed. J. Laird) Morgan Kaufmann.

17

Varying Levels of Abstraction in Qualitative Modelling

I. Mozetič†
J. Stefan Institute

I. Bratko
E. Kardelj University and J. Stefan Institute

T. Urbančič
J. Stefan Institute

Abstract

We describe a formalism for hierarchically representing qualitative models at various levels of abstraction. The formalism is based on logic, namely on typed Horn clauses also known as database clauses. The notion of abstraction is realized through a hierarchy of types for the domains of predicates. The abstraction hierarchy can be used in generating explanations with an adjustable degree of detail; also in improving search efficiency in solving tasks of diagnosis and control, as well as the learning of qualitative models. Results obtained at a simpler, more abstract level, can be used to guide the search at a more detailed and combinatorially more complex level. The corresponding algorithms are presented as PROLOG programs and their behaviour studied on example problems including a qualitative model of the heart.

1. INTRODUCTION

Deep knowledge which takes into account underlying principles of a problem domain enables 'reasoning from first principles'. This helps to alleviate the knowledge acquisition bottleneck, improves robustness, achieves clearer semantics and better explanation capabilities of expert systems (e.g. Steels 1985). In this respect, knowledge representation in the form of a qualitative model is particularly important.

In this paper we investigate a representation formalism with a particular view on qualitative modelling. In our approach, a model is defined by its structure (a set of components and their connections) and the functions of its constituent components. The model may be in different qualitative states (defined by the states of the components) which

†Present address: Austrian Research Institute for Artificial Intelligence, Vienna, Austria.

indicate regions where some laws of behaviour are valid. The model accepts some input and, depending on the state, transforms it into output. For example, a model of a pressure regulator (as defined in de Kleer and Brown 1984) consists of a valve and a pressure sensor. The state of the regulator is defined solely by the state of the valve which is either closed, working, or open. The values of the air pressure and flow at input are transformed into the output values, depending on the qualitative state of the regulator. In this paper we are concerned only with the behaviour of a model within a qualitative state. We do not consider possible transitions between different states over a period of time, usually called 'envisioning' (de Kleer and Brown 1984).

A model of a device can be used in various tasks, such as simulation, diagnosis, or control. Formally, the simulation task is to derive an output from a given input and state of the model. The diagnostic task is the opposite: given an input and an output, find all possible states that may produce the given behaviour. This assumes that the presence of faults in a system is modelled through its internal states. Models are usually not designed directly for diagnosis and are therefore often unsuitable for direct use for this purpose; typically, the problem with diagnostic reasoning is its combinatorial complexity.

The simplest method of using a model for diagnosis is based on the 'generate and test' problem-solving strategy. This method generates potential solutions (possible states) and tests for given input–output behaviour. This method is very inefficient if the number of possible states is large, e.g. in the case of multiple faults.

In the paper we suggest two methods to improve the diagnostic efficiency as compared with the naïve 'generate and test' approach. The first method represents a model on several levels of abstraction, and takes advantage of hierarchies to reduce the space of potential solutions. The problem is first solved at a simpler, more abstract level. This coarse solution is then used to guide the search at more detailed levels. The abstraction hierarchy also provides better explanation and offers a trade-off between the diagnostic time and the specificity of diagnoses. The second method takes advantage of knowledge representation based on logic which enables direct execution of a model in the 'backward' direction.

Both methods were developed, implemented, and successfully tested in the domain of the KARDIO expert system (Bratko, Mozetic, and Lavrac 1989). The task in question was the diagnosis of cardiac arrhythmias from electrocardiographic (ECG) descriptions. A brief overview is given below. An expert system for ECG diagnosis was under development based on rules directly relating arrhythmias to the corresponding ECG descriptions (Lavrac *et al.* 1985). However, due to the combinatorial nature of multiple disorders that may occur simultaneously in the heart,

we were not able to construct a complete knowledge base. For that reason, we have manually developed a qualitative model of the heart that simulates its electrical activity. The model was used for automatic derivation of a complete rule base, relating 2419 heart disorders to 140,969 ECG descriptions (Bratko, Mozetic, and Lavrac 1989). Owing to its space complexity this knowledge base is awkward for direct use. It was therefore compressed by means of 'learning-from-examples' techniques into an operational base of diagnostic rules approximately 30 times smaller (Mozetic 1986). Here, the model was used for diagnosis in an indirect way. The next step was to make deep knowledge, represented as a qualitative model, operational by itself. The idea was to achieve operationality by introducing abstraction hierarchy into the model. For the sake of this research, we have used a small, but still complicated subset of this complex diagnostic domain. Most of the material presented here is adapted from Mozetic (1988).

In the following section, we define a logic-based formalism for representing single-level and hierarchical qualitative models. Section 3 describes model interpretation with emphasis on the use of a hierarchical model for diagnosis. In Section 4 experiments and results using the model of the heart are presented. Section 5 demonstrates how the idea of hierarchical diagnosis can be used for solving equations. Conclusions are set out in Section 6.

2. REPRESENTING QUALITATIVE MODELS

As opposed to the process-oriented approach to qualitative modelling (Forbus 1984) we take the component-oriented approach. We represent a model as a structure, defined by a set of components and their interconnections. The model's behaviour is defined solely by its structure and the behaviour of its components (de Kleer and Brown 1984). A model has an internal qualitative state and is connected to the environment. The state of the model is defined by qualitative states of its components and denotes a range where some laws of behaviour are valid. A model accepts an input from its environment, and transforms it to the output, depending on its qualitative state.

Representation in the system is based on the deductive hierarchical database formalism (Mozetic 1987a). A qualitative model consists of a hierarchy of non-recursive database clauses. A database clause is a typed Horn clause, augmented with negation that may be used in the body of the clause (Lloyd and Topor 1985). Database clauses are typed in that the domains for predicate arguments are defined. Domains are sets of (possibly structured) values. Domains are finite which helps to implement negation correctly (unlike standard PROLOG) in the body of a clause. The set of clauses must be non-recursive where indirect (cyclic)

recursion is not allowed either. This non-recursive property is referred to as hierarchy by Lloyd and Topor (1985). In this paper, however, hierarchy refers to levels of abstraction. The interpreter used is an extension of the standard PROLOG interpreter (Lloyd 1984) that correctly handles negation, types, and different goal selection strategies. In the following paragraphs we look first at representing a model at a single level, and then introduce the abstraction hierarchies.

2.1. One-level representation

A single-level qualitative model is defined by:

(1) the structure of the model;

(2) functions of its constituent components;

(3) utility predicates.

The structure is defined by a set of components and their connections. The structure can be viewed as an undirected graph. It is represented by a database clause. The head of the clause denotes a relation between a qualitative state of the model, an input, and an output from the model. In the body of the clause (a conjunction of literals), each literal corresponds to a component of the model, and shared variables between literals represent connections between components. The qualitative state can be used to model, for example, different operational regions of a device (e.g. a closed valve or an open valve), or internal faults under which the behaviour of the device changes. The literals in the body of the clause are assumed to be ordered in a way that reflects the propagation of input parameters throughout the graph towards the outputs. This ordering can be associated with causal chains of events. However, such an ordering is in our formalism only a convenience and is not necessary in principle. For some systems, where signals propagate cyclically in all directions, there is no such natural input–output ordering. In such cases several orderings are equally suitable.

As an example we take a model of the pressure regulator (de Kleer and Brown 1984). The regulator consists of a valve and a pressure sensor (Figure 1). Variables Q, P, and X denote changes in the air flow, pressure, and the valve area, respectively. As in de Kleer and Brown (1984) we choose the ordered set $\{-, 0, +\}$ as the quantity space for these variables. Table 1 gives the laws for qualitative addition and negation on this quantity space.

The structure of the model is defined by two components (valve and sensor) and their interconnections:

```
regulator( State, Pi, Qi, Po, Qo ) ←
    valve( State, Pi, Qi, X, Po, Qo ),
    sensor( Po, Qo, X ).
```

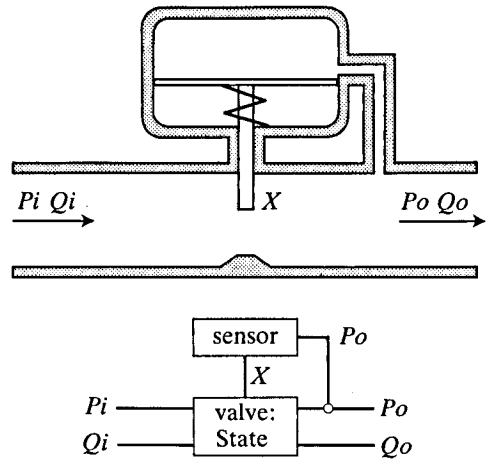


Figure 1. A pressure regulator. Variables *Pi* and *Qi* denote changes in the input pressure and flow, *X* denotes change in the valve area, and *Po* and *Qo* denote changes in the output pressure and flow, respectively. The bottom diagram represents the structure of the regulator, i.e. a set of components and their interconnections.

Table 1. Qualitative addition and negation.

X Y+	−	0	+	X	−X
−	−	−	−,0,+	−	+
0	−	0	+	0	0
+	−,0,+	+	+	+	−

The qualitative state of the regulator is defined only by the state of the valve which may be 'open', 'closed', or 'working'. It depends on the area for flow in the valve: the valve is either completely open (state 'open') or completely closed (no flow, state 'closed') or between (state 'working'). If the valve is completely closed there is no flow change (and no flow either):

valve(closed, Pi, Qi, X, Po, Qo) ←
  Qi = 0, % no change in in-flow
  Qo = 0. % no change in out-flow

If the valve is completely open then any change in pressure or flow at the output is the same as the change at the input:

valve(open, Pi, Qi, X, Po, Qo) ←
  Pi = Po,
  Qi = Qo.

However, if the valve is 'working' (between closed and open), then its

behaviour is determined by the following relations between the variables:

```
valve( working, Pi, Qi, X, Po, Qo ) ←
  Pi − Po = Pio,   % Pio is a drop of pressure from input to output
  Pio + X = Qi,    % in-flow is proportional to the pressure
                   % drop and area of the valve
  Qi = Qo,         % conservation of flow
  Qo = Po.         % out-flow is qualitatively proportional to
                     pressure
```

The sensor takes care that any change in the area of the valve is inversely proportional to the change in the output pressure:

```
sensor( Po, Qo, X ) ←
  X = − Po.
```

The definition of types of predicate argument could be as follows:

```
type( regulator(state, quantity, quantity, quantity, quantity) ).
type( valve(state, quantity, quantity, quantity, quantity, quantity) ).
type( sensor(quantity, quantity, quantity) ).
domain( state, [closed, working, open] ).
domain( quantity, [ −, 0, + ] ).
```

A qualitative simulation yields the model's behaviour under given conditions. For example, assume that the pressure regulator is in the state 'working' and that the input pressure is increasing (Pi = +). Then the results of simulation are the following variable values which satisfy the above relations:

```
Qi  = +
Qo  = +
X   = −
Po  = +
Pio = +
```

The interpretation of these results reveals that the output pressure and flow (Po, Qo) are increasing, while the valve area (X) is decreasing. The result is perhaps not quite as expected, namely, that the output pressure is constant. However, the simulation shows that the input pressure is increasing faster than the output pressure, since their difference (Pio) is increasing.

This simple example also demonstrates some advantages of a deep model with respect to 'surface models'. A model makes possible a causal explanation of the device's behaviour (e.g. the pressure affects the valve area, and the latter affects the flow through the valve). Further, a model may contain meaningful concepts which become meaningless when the

structure of the model is unknown (e.g. the area of the valve). In the case of the pressure regulator, if we omit the flow, the following are examples of two 'surface' rules:

State = working, Pi = + ⇒ Po = +
State = working, Pi = 0 ⇒ Po = 0

The first rule states that if the regulator is 'working' and the intput pressure is increasing then the output pressure is increasing as well. However, the rule does not explain why this is so, nor that the input pressure is increasing faster than the output pressure.

Notice that the regulator model in the foregoing example is a single-level model although the regulator is composed of a valve and a sensor. We consider abstraction hiearchy in the next section.

2.2. Hierarchy of models

It is useful to represent a qualitative model at several levels of abstraction. In this case a model consists of a hierarchy of single-level models, where the top model is the most abstract (also the simplest), and the bottom model in the hierarchy is the most detailed (also the most complicated one). A hierarchy of models improves the efficiency of model interpretation when we use the model for diagnosis, and may also improve the explanation capabilities of the system.

As an example let us consider a very abstract model of the electrical activity in the heart. From this extremely abstract view, the heart is simply a box which consists of a generator of electrical impulses (giving the pace of the heart), and these impulses cause, through another component, some external manifestation, called ECG (see Figure 2).

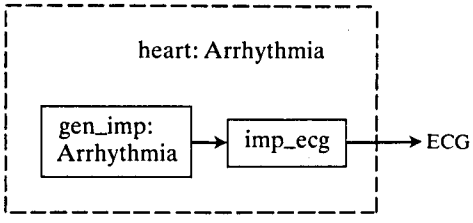


Figure 2. A very abstract model of the heart.

The heart can suffer from an arrhythmia which can be modelled as an internal state of the generator of impulses which affects its behaviour. The model consists of only two components: a generator of impulses (gen_imp) and a generator of ECG descriptions (imp_ecg). The model has no input and relates any state of the heart (Arrhythmia) to the output (ECG):

```
heart( Arrhythmia, ECG ) ←
  gen_imp( Arrhythmia, Impulse ),
  imp_ecg( Impulse, ECG ).
```

The function of the impulse generator may be represented by two ground unit clauses:

```
gen_imp( slow_rhythm, under_60 ).
gen_imp( fast_rhythm, over_60 ).
```

A definition of the generator of ECG descriptions completes the example. Note that here 'Rate' denotes a universally quantified variable:

```
imp_ecg( Rate, Rate ).
```

The above model of the heart is extremely simple. If the user wants to push the level of detail in representation, the system allows him to:

(1) specify more detailed structure, replacing a component in the model by a set of components (Figure 3);

(2) refine values of variables by defining hierarchies of values (Figure 4);

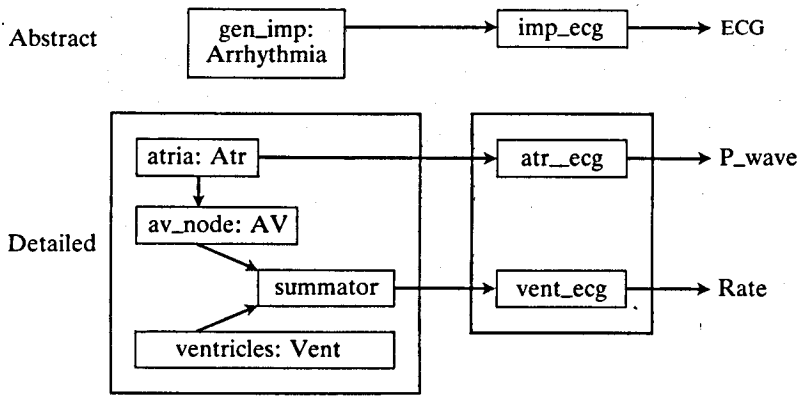


Figure 3. Refinement of the model's structure.

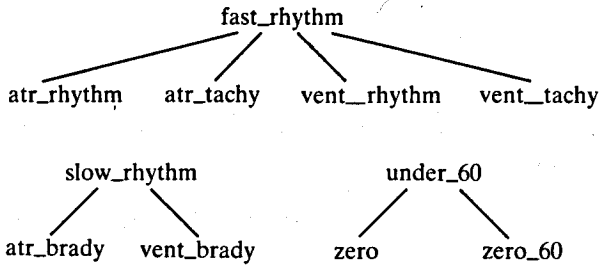


Figure 4. Examples of hierarchies.

(3) introduce new variables, not relevant at the more abstract level.

In the more detailed model shown in Figure 3, the heart consists of the atria, the ventricles, and the AV (atrio-ventricular) node which conducts impulses from the atria to the ventricles. Further, in the ECG, not only is the rate of heartbeats of interest, but the presence of the P-wave as well.

The more detailed structure of the model is represented by the following clause:

```
heart( arr(Atr,AV,Vent), ecg(P_wave, Rate)) ←                    (1)
   atria( Atr, ImpAtr ),
   av_node( AV, ImpAtr, ImpHis ),
   ventricles( Vent, ImpVent0 ),
   summator( ImpVent0, ImpHis, ImpVent ),
   atr_ecg( ImpAtr, P_wave ),
   vent_ecg( ImpVent, Rate ).
```

Types of arguments for all predicates in the model definition must be specified. Each type defines a set of terms, i.e. the domain of some variable. A term is either simple (a constant) or compound (a functor applied to a number of arguments). For example, at the abstract level, there are only two possible states (simple arrhythmias), while at the detailed level the state of the heart is defined by the states of the atria, the AV node and the ventricles:

```
type( heart( arr, ecg )).
```

Abstract:

```
domain( arr, [slow_rhythm, fast_rhythm] ).
```

Detailed:

```
domain( arr, arr(atr, av, vent) ).
domain( atr, [quiet,atr_brady,atr_rhythm,atr_tachy] ).
domain( av, [normal, av_block] ).
domain( vent, [quiet,vent_brady,vent_rhythm,vent_tachy] ).
```

Hierarchies of terms specify relations between values on the abstract and detailed levels. Note that the abstract level is incomplete with respect to the detailed level in our case, since it does not contain it properly (e.g. there is no notion of impulse conduction on the abstract level). However, it is required that the detailed level is consistent with the abstract level (Mozetic 1990).

The definition of the detailed model is completed when functions of its components are defined. In our case we have:

```
atria( quiet,        zero ).
atria( atr_brady,    zero_60 ).
atria( atr_rhythm,   60_100 ).                                    (2)
atria( atr_tachy,    over_100 ).

av_node( normal,    Rate1, Rate1 ).                               (3)
av_node( av_block, Rate1, Rate1 ).                                (4)
av_node( av_block, Rate1, Rate2 ) ←                               (5)
  succ( Rate2, Rate1 ),
  Rate2 := zero_60 v 60_100.

ventricles( quiet,          zero ).
ventricles( vent_brady,     zero_60 ).                            (6)
ventricles( vent_rhythm,    60_100 ).                             (7)
ventricles( vent_tachy,     over_100 ).

summator(zero, zero, zero).
summator(zero, Rate, Rate) ← ~(Rate = zero).                      (8)
summator(Rate, zero, zero) ← ~(Rate = zero).                      (9)

atr_ecg( Rate, present ) ← ~(Rate = zero).                        (10)
atr_ecg( zero, absent ).

vent_ecg( Rate, Rate ).                                           (11)
```

We can specify some background knowledge in the form of utility predicates that may be used in a component definition. For example, the following utility predicate defines the ordering of rates:

```
succ( zero,      zero_60 ).
succ( zero_60,   60_100 ).
succ( 60_100,    over_100 ).
```

3. INTERPRETING QUALITATIVE MODELS

The representation of a model based on logic has an important property in that it involves relations. In computation, relations can be used in any direction. If X and Y are in some relation r, written in the model as $r(X,Y)$, one can determine Y given X, and X given Y as well. For that reason, the same relation-based model can be used in both directions: for 'forward' and 'backward' reasoning. Forward reasoning is realized with the left-to-right goal selection strategy (as in standard PROLOG) and is usually suitable for simulation. Backward reasoning is realized with the right-to-left goal selection strategy and can be used for diagnosis.

3.1. Hierarchical diagnosis

Hierarchical diagnosis is based on the representation of a model on

several levels of abstraction. The basic idea behind the method is to move the 'greedy' search from a detailed to an abstract level of the model. Diagnosis at the abstract level is less precise, but easier because of the simplicity of the model at this level. More detailed diagnoses are obtained through simulation at the detailed level. An illustration of the method is given in Figure 5. This illustration is concerned with the particular diagnostic task of finding arrhythmias for a given electrocardiogram.

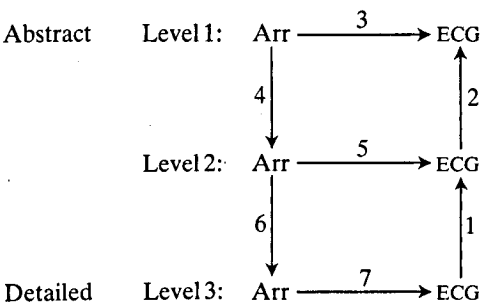


Figure 5. A scheme of hierarchical diagnosis.

Given a detailed ECG, the diagnostic algorithm climbs the hierarchies of values to find a more abstract ECG (steps 1, 2). The algorithm uses the model at the abstract level to find an arrhythmia that can actually produce the abstract ECG (step 3). If more detailed diagnosis is required, the algorithm uses hierarchies to consider further only arrhythmias that are below the abstract arrhythmia in the hierarchy (steps 4, 6). Potentially possible detailed arrhythmias are then verified by means of simulation to check that they actually produce the corresponding ECG (steps 5, 7). If another diagnosis is required, the algorithm backtracks, until there are no more possible arrhythmias.

Our model representation formalism allows for an abstract level to be incomplete with respect to the detailed level. In such a case the algorithm cannot take the advantage of hierarchies. In the degenerate case of total incompleteness, the method is in fact reduced to the naïve 'generate and test' strategy.

The diagnostic algorithm below is written in standard PROLOG. For a given level of the model and output from the model the procedure 'diagnose(Level, Out, In)' finds a corresponding input of the model (In). The problem is formally the same when, for a given output, either an input or an internal state is to be found. The procedure consists of two clauses. The first clause deals with the case when there are hierarchies, and the second clause deals with all instances of inputs or states that do not have abstract counterparts at the given level of abstraction:

```
diagnose( Level, Out, In ) ←
  Level > 1, Level0 is Level − 1,
  type( model(InType, OutType) ),
  hierarchy( Level0, OutType, Out0, Out ), % level up
  diagnose( Level0, Out0, In0 ),
  hierarchy( Level0, InType, In0, In ),        % level down
  verify( Level, In, Out ).
diagnose( Level, Out, In ) ←
  no_abstract( Level, In ), % there is no abstraction
  verify( Level, In, Out ).
```

The procedure 'verify' verifies whether a given input (or state) really causes a given output on a specified level of abstraction. In the simplest case, the procedure may be equivalent to the simulation algorithm:

```
verify( Level, In, Out ) ←
  simulate( Level, In, Out ).
```

Instances of inputs or states that do not have any abstractions may be either explicitly stated or can be computed, for example:

```
no_abstract( Level, In ) ←
  Level0 is Level − 1,
  type( model(InType, _) ),
  not hierarchy( Level0, InType, _, In ).
```

The following example illustrates the method. The model under consideration is our previous detailed model of the heart. The task is to find an arrhythmia, given the following ECG description: 'P_wave' is 'present', 'Rate' is 'zero_60':

```
Detailed ECG = ecg( present, zero_60 )
```

Given a detailed ECG, the diagnostic algorithm climbs the hierarchies of values (Figure 4) to find more abstract ECG's:

```
Abstract ECG = under_60
```

The algorithm uses the model at the abstract level to find an arrhythmia that actually produces this particular abstract ECG. The following arrhythmia is obtained by means of the 'generate and test' method:

```
Abstract Arr = slow_rhythm
```

To find more detailed diagnoses, the algorithm uses hierarchies to consider further only arrhythmias that are in the hierarchy under the abstract arrhythmia 'slow_rhythm' (Figure 4) and satisfy given constraints over the states of the heart. Constraints specify which

(multiple) arrhythmias are physiologically possible and medically interesting. Potentially possible detailed arrhythmias are then verified by means of simulation:

```
Detailed Arr =
  arr( atr_brady, normal, quiet ) SUCCEEDS
  arr( quiet, normal, vent_brady ) FAILS
```

The following arrhythmias are not considered at all since they are not under the hierarchy of 'slow_rhythm':

```
Detailed Arr =
  arr( atr_rhythm, normal, quiet )
  arr( atr_tachy, normal, quiet )
  arr( quiet, normal, vent_rhythm )
  arr( quiet, normal, vent_tachy )
```

However, the abstract level is incomplete, meaning that there are some detailed arrhythmias that do not have corresponding abstractions:

```
no_abstract( 2, arr(_, av_block, _) )
```

They have to be explicitly verified:

```
Detailed Arr =
  arr( atr_brady, av_block, quiet ) SUCCEEDS
  arr( atr_rhythm, av_block, quiet ) SUCCEEDS
  arr( atr_tachy, av_block, quiet ) FAILS
```

As compared to naïve diagnosis, in this example the search space was reduced by almost one half.

3.2. Backward diagnosis

As already mentioned, the standard PROLOG interpreter uses a left-to-right goal selection strategy. In the case of diagnosis, the output is given and the state of the model has to be found. The use of components as constraints in the inverse direction better suits this problem. In logic programming terminology, this is the right-to-left goal selection strategy.

To illustrate this method, let us consider the same example as in the previous section. Find an arrythmia, given:

```
ECG = ecg(present, zero_60)
```

The interpreter is given the following goal

```
heart( Arr, ecg(present, zero_60) )
```

in order to find an arrhythmia that causes the given ECG. The interpreter unifies the goal with the head of clause 1 and replaces the goal with the body of clause 1. Since the right-to-left goal selection strategy is used, the last (rightmost) subgoal of the clause 1 is selected next. The following is a trace of the first (unsuccessful) derivation:

```
vent_ecg( zero_60, zero_60 )
atr_ecg( over_100, present )
summator( zero, zero_60, zero_60 )
ventricles( quiet, zero )
av_node( av_block, over_100, zero_60 )
FAILURE
```

The last subgoal cannot be satisfied since it cannot be unified with the heads of clauses 3 and 4, and the condition in the body of clause 5 cannot be satisfied either. However, the reason for failure is actually in clause 10, since for the second subgoal 'atr_ecg(Rate, present)', the interpreter has to make a non-deterministic choice for the variable Rate, other than 'zero'. The first choice above ('over_100') leads to failure, therefore the interpreter backtracks to the last point of non-determinism and tries the next possible choice:

```
summator( zero_60, zero, zero_60 )
ventricles( vent_brady, zero_60 )
av_node( av_block, over_100, zero )
FAILURE
```

Again, failure occurs since clause 9 was used instead of clause 8. The system backtracks again and chooses the next possible value for 'Rate' which now gets the value '60_100'. Now, the derivation is successful:

```
atr_ecg( 60_100, present )
summator( zero, zero_60, zero_60 )
ventricles( quiet, zero )
av_node( av_block, 60_100, zero_60 )
atria( atr_rhythm, 60_100 )
SUCCESS
```

Since the initial goal was satisfied, the interpreter returns the substitution of the free variable (Arr) in the question:

```
Arr = arr(atr_rhythm, av_block, quiet)
```

Two other possible diagnoses are obtained through backtracking in the same manner:

```
Arr = arr(atr_brady, normal, quiet)
Arr = arr(atr_brady, av_block, quiet)
```

4. EXPERIMENTS AND RESULTS

The model representation and diagnostic methods described in this paper were used in an implementation of a subset of the KARDIO model for ECG diagnosis. The system embodies a model of the heart, represented on three levels of abstraction. Table 2 gives the complexity of the model at each level, with respect to the number of its constituent components, the number of possible states of the model (arrhythmias), and outputs from the model (ECG descriptions). The fourth column in the table indicates the number of arrhythmias that do not have a corresponding abstraction at the higher level, i.e. that are unique to the model at the detailed level.

Table 2. Complexity of the heart model used in the experiments.

Level of abstract	Components	States (Arr) All	States (Arr) No abstract	Outputs (ECG)
1	2	3	3	3
2	9	18	3	38
3	16	175	85	333

4.1. An example of hierarchical diagnosis

Hierarchical diagnosis offers a tradeoff between the specificity of diagnoses, certainty (the number of alternatives), and the time one is willing to wait for an answer. This is closely related to the idea of Variable Precision Logic (Michalski and Winston 1986).

A detailed ECG description at the third level of abstraction is given below:

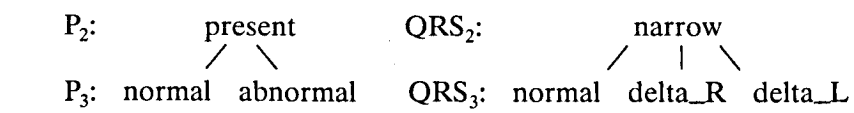
Ecg_3 = [P = abnormal, Rate_P = b_100_250, P_QRS = always_QRS,
PR = shortened, Rhythm = regular, QRS = normal,
Rate_QRS = b_100_250]

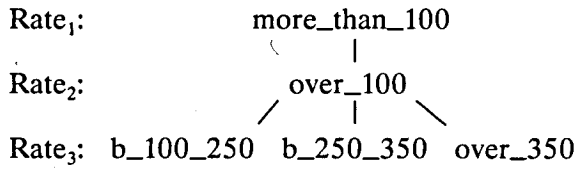
The user is interested in all possible diagnoses. The hierarchical diagnostics algorithm first finds corresponding abstract ECG descriptions (at the second and first level of abstraction):

Ecg_2 = [P = present, P_QRS = always_QRS, QRS = narrow,
Rate = over_100]

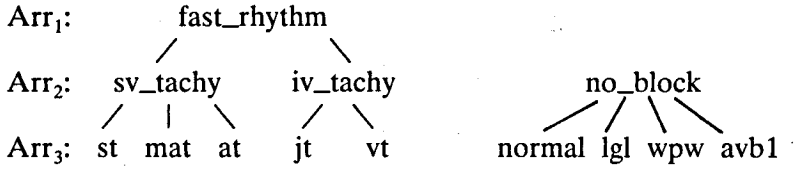
Ecg_1 = [Rate = more_than_100]

The following ECG hierarchies are used:





The algorithm uses the model of the heart represented on all three levels of abstraction and the following hierarchies of arrhythmias:



When interacting with the system the user first gets abstract diagnoses, and if the time allows, these are refined in more detail and more alternative diagnoses are produced. In the following (paraphrased) dialogue with the system, the user responses are underlined. The essential information in the system's diagnoses is in bold typeface, and these essential pieces of information are expanded into corresponding medical terms.

Possible diagnoses:
> Arr_1 = **fast_rhythm**
tachycardia

Do you want a more detailed diagnosis? yes
> > Arr_2 = arr(**sv_tachy**, no_block, quiet)
supra-ventricular tachycardia

Do you want a more detailed diagnosis? yes
> > > Arr_3 = arr(sv(quiet, **at**), normal, iv(quiet, normal, quiet))
atrial tachycardia

Do you want an alternative diagnosis? yes
> > > Arr_3 = arr(sv(quiet, **at**), **lgl**, iv(quiet, normal, quiet))
atrial tachycardia with LGL syndrome

Do you want an alternative diagnosis? yes
> > Arr_2 = arr(quiet, no_block, **iv_tachy**)
intra-ventricular tachycardia

Do you want a more detailed diagnosis? yes
> > > Arr_3 = arr(sv(quiet, quiet), normal, iv(**jt**, normal, quiet))
junctional tachycardia

Do you want an alternative diagnosis? yes
There are no more possible diagnoses!

4.2. Diagnostic efficiency

Experiments to compare the efficiency of different diagnostic methods were carried out with the model at the third level of abstraction. We compared:

(1) the naïve 'generate and test' method;

(2) hierarchical diagnosis with simulation;

(3) one-level backward diagnosis;

(4) a combined method using hierarchical diagnosis (when there are abstractions) and backward diagnosis (for arrhythmias that do not have corresponding abstraction).

A randomly selected subset of 12 ECG descriptions was used to measure the time that each method needed to find possible arrhythmias. Table 3 gives average times needed to find the first and the rest of possible arrhythmias. The analysis of results revealed that times were dependent on whether the possible arrhythmia has an abstraction or not. Therefore, four more columns are included in Table 3, two for each case. The first column in each category indicates the time needed to find the first arrhythmia, and the second column indicates the further time that was spent until all possibilities were exhausted.

The results show that the naïve 'generate and test' diagnosis can be considerably improved. Hierarchical diagnosis is much faster when it can take the advantage of abstractions (consider the 5th column, 32 sec. vs. 313 sec.). Even when the arrhythmia that is actually possible does not have the abstraction, the search space is reduced (3rd and 4th column together, 460 sec. vs 802 sec.). Backward diagnosis is the most efficient in this model, but mostly due to the relatively large number of specific diagnoses without abstractions. When there are abstractions, hierarchical diagnosis with simulation is not much slower than backwards diagnosis.

Table 3. Average times (in CPU seconds on VAX 11/750) for different diagnostic methods implemented in interpreted PROLOG.

Diagnostic method	Category 1 Together (12 cases)		Category 2 No abstract (3 cases)		Category 3 Abstract (9 cases)	
	First	Rest	First	Rest	First	Rest
Naïve	305	500	283	519	313	493
Hierarchical	108	408	335	125	32	502
Backward	50	34	160	40	14	32
Combined	68	92	172	44	33	108

Here it should be stressed that relatively large absolute times needed for diagnosis (a few minutes) are not inherent to the problem. They are due to the inefficient model interpreter (written in C-PROLOG interpreter) and could be reduced at least by a factor of 100 with compiled PROLOG. Results of recent experiments with compiled Quintus Prolog and the heart model represented at 4 levels of abstraction (Mozetic 1990) show a speed-up of a factor of 200.

5. ANOTHER APPLICATION: SOLVING EQUATIONS

The algorithm 'diagnose' described in Section 3.1 is general. Although it was developed in order to solve the task of diagnosis it can be used without change for other purposes also. Here we show how it can be used for solving equations where it is not possible to find the exact analytical solution. Among numerical methods for finding approximate solutions, the method of bisection is well known. Its main virtue is its simplicity, although there are other numerical methods that are more efficient. It is interesting that our hierarchical diagnostic algorithm, when applied to numerical solving of equations, emulates the method of bisection.

A function

$$Y = f(X)$$

can be viewed as a special kind of model. The input is the value of the independent variable X. The model transforms it into the output, namely into the value of the variable Y. Obviously the notion of state can be neglected here. Consider the equation

$$f(X) = y$$

To solve it means finding the input which gives the output value y. In this sense solving equations is similar to the task of diagnosis.

Let us consider a continuous function $Y = f(X)$ on a given interval $[e1,e2]$. Let us have a partition P of $[e1,e2]$ into subintervals which can intersect only in divisional points and cover the whole interval $[e1,e2]$. The function can then be represented as a relation

$$\text{model}(X,Y)$$

which is true if $X = X\text{Int}$ is one of the subintervals in the partition P of $[e1,e2]$ and $Y = [y1,y2]$ such that

$$y1 = \min_{x \in X\text{Int}} f(X) \quad \text{and} \quad y2 = \max_{x \in X\text{Int}} f(X)$$

There is an obvious correlation between the number of subintervals and the degree of accuracy to which we can specify the value of y given X (and vice versa). The partition P is defined by two parameters K and N

and is obtained from the initial interval by N steps of K-section where K-section is the division of an interval into K equal subintervals. Accordingly, P consists of K^N intervals.

Thus the model has two parameters X and Y whose values are intervals. The 'model' relation has to be defined with regard to the function f. As an example the function

$$Y = X + \tan(X)$$

on the interval [0,1] will be considered. The relation 'model' can be defined, using interval arithmetic, in PROLOG as follows:

```
model( X, Y ) :-
  tan_int( X, TanX ),
  add_int( X, TanX, Y ).

% tangent of an interval which does not contain any pole
% A .. B means interval [A,B]
tan_int( X 1 .. X 2, TanX 1 .. TanX 2) :-
  TanX 1 is tan( X 1),
  TanX 2 is tan( X 2).

% addition of intervals
add_int( X1 .. X2, Y1 .. Y2, Z1 .. Z2) :-
  Z1 is X1 + Y1,
  X2 is X2 + Y2.

% type( model( InType, OutType)).
type( model( x, y)).
```

The definition is rather simple because the functions involved are monotonic.

Of the three ways for refining the level of detail of a model mentioned in Section 2.2, here we use only one: refining values of variables by defining hierarchies of values. Hierarchies for X are defined with a simple rule: The interval $[A,B]$ is in the hierarchy below the interval $[X,Y]$ if $[A,B]$ is one of the intervals obtained by the division of $[X,Y]$ into K equal intervals. For Y we have a trivial hierarchy, as the domains are the same at all levels.

Total completeness is ensured by construction. The initial interval (i.e. the value of X at the first level) is the only one without abstraction:

```
% no_abstract( Level, In).
no_abstract( 1, 0 .. 1).
```

The model on the i-th level differs from the model on the $(i$-1)-th level only in the interval values of X. On the i-th step of K-section each of the values of X on the $(i$-1)-th level is divided into K subintervals. In such a way all the values on the i-th level are obtained.

The task of finding a solution of an equation $f(X) = y$ can now be stated as follows; given a real number y find an interval XInt such that the relation 'model(XInt, YInt)' is satisfied and y is in YInt.

```
% verify( Level, X, y).
verify( _, In, Out) :—
  model( In, Y),
  is_in( Out, Y).
```

Now we can apply the 'diagnose' algorithm to our task of finding the solution of an equation. As an example let us find the solution of the equation

$$X + \tan(X) = 1$$

on the interval [0, 1] (see Figure 6).

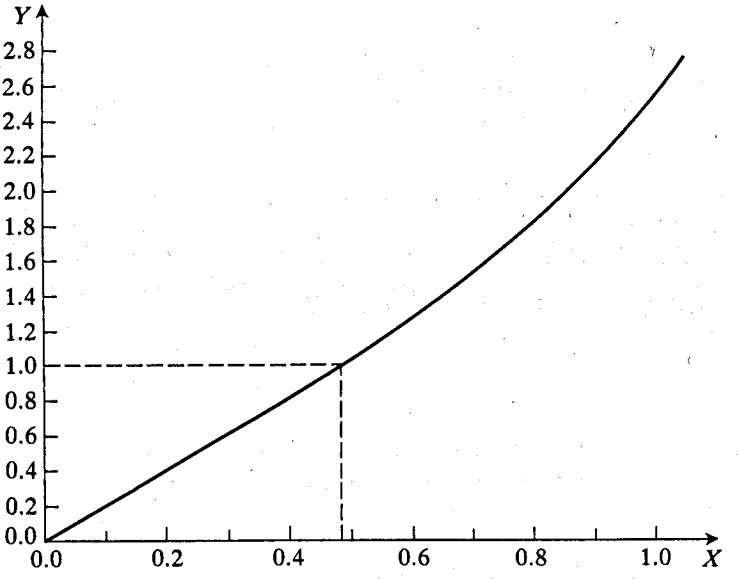


Figure 6. Graph of the function $Y = X + \tan(X)$.

We have to specify the parameter K. For example, $K = 2$ determines that we will look for a solution with the method of bisection. Now we only have to instantiate the values of Level and Out. In our case Out has value 1. So the solution on the 10-th level is found by:

```
?- diagnose( 10, 1, X).
X = 0.47851563 .. 0.48046875
```

At the 30-th level the solution accurate to eight decimal places is obtained:

```
?- diagnose( 30, 1, X)
X = 0.47973101 .. 0.47973101
```

Some interesting results are collected in Table 4.

Table 4. Levels and times needed for finding X accurate to 8 decimal places using bisection, 10-section, and 100-section (interpreted PROLOG on IBM AT).

K	Level	Time (seconds)
2	30	1.70
10	10	2.03
100	6	10.77

6. CONCLUSION

Often it is difficult or impossible to execute a model in the backward direction. In this respect, representation at multiple levels of abstraction and the use of hierarchical diagnosis seems promising. A representation formalism with abstraction hierarchy based on logic was described, together with the corresponding hierarchical diagnostic algorithm. This can also be used for backward execution of a model or inverse simulation.

The method was applied to a hierarchical model of the heart and ECG diagnosis based on this model. Another exercise in using this algorithm in equation-solving indicates the applicability of the method as a general technique of logic programming.

Further research centred around the proposed representation is concerned with automatic learning in the context of qualitative modelling, aiming at machine-aided construction of qualitative models (Mozetic 1987a, b). The learning system developed has already been successfully applied in interactive construction of a model of the heart, given the structure of the model and some examples of its behaviour (Mozetic 1988).

REFERENCES

Bratko, I., Mozetic, I., and Lavrac, N. (1989). KARDIO—a study in deep and qualitative knowledge for expert systems. MIT Press, Cambridge, MA.

de Kleer, J. and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence* **24**, Nos 1–3, pp. 7–83.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence* **24**, Nos 1–3, p. 168.

Lavrac, N., Bratko, I., Mozetic, I., Cercek, B., Grad, A., and Horvat, M. (1985). KARDIO-E—an expert system for electrocardiographic diagnosis of cardiac arrhythmias. *Expert Systems* **2** No. 1.

Lloyd, J. W. (1984). *Foundations of logic programming.* Springer-Verlag.

Lloyd, J. W. and Topor, R. W. (1985). A basis for Deductive Database Systems. Journal of Logic Programming **2** No. 2, pp. 93–109.

Michalski, R. S. and Winston, P. (1986). Variable Precision Logic. *Artificial Intelligence* **29**, No. 2, pp. 121–46.

Mozetic, I. (1986). Knowledge extraction through learning from examples. In *Machine Learning: A Guide to Current Research* (Eds T. M. Mitchell, J. G. Carbonell, and R. S. Michalski), Kluwer Academic Publishers, Boston.

Mozetic, I. (1987a). Learning of qualitative models. In *Progress in Machine Learning* (Eds I. Bratko and N. Lavrac), Sigma Press, Wilmslow, UK.

Mozetic, I. (1987b). The role of abstractions in learning qualitative models. In *Proc. of the Fourth International Workshop on Machine Learning,* Morgan Kaufmann, Irvine, CA, June 22–25.

Mozetic, I. (1988). Automatic construction of qualitative models. Ph.D. Thesis (in Slovenian), E. Kardelj University, Ljubljana, Yugoslavia.

Mozetic, I. (1990). Diagnostic efficiency of deep and surface knowledge in KARDIO. *Artificial Intelligence in Medicine,* **2**, No. 2, 67–83.

Steels, L. (1985). Second generation expert systems. *Future Generation Computer Systems* **1**, No. 4, pp. 213–21.

APPLICATIONS AND MODELS OF KNOWLEDGE ACQUISITION

18

Information Content of Chess Positions: Implications for Game-Specific Knowledge of Chess Players

J. Nievergelt†
Department of Computer Science,
University of Illinois, USA

1. INTRODUCTION

Chess has served as a convenient vehicle for studying cognition and perception (see de Groot 1965, and Chase and Simon 1973) as well as machine intelligence. Perhaps *the* central question for both of these research uses of chess is: How much chess-specific knowledge does it take to play at a given level of competence, for example, at the master level? It is difficult to say what chess-specific knowledge is, and it certainly consists of different types of knowledge that must be considered independently of each other (for example, 'book knowledge' is very different from experience obtained in over-the-board play). Even if one succeeds in defining what 'chess-specific knowledge' is, there remains the difficulty of measuring it. Because of these difficulties, any approach to measuring the amount of knowledge possessed by a practitioner of a craft must be based on questionable assumptions, and any result obtained is subject to uncertainty and criticism. Only the inherent interest of the question posed justifies reporting on a rough and inconclusive experiment designed to answer one aspect of the tantalizing question: How much chess-specific knowledge does it take to play at a given level of competence?

2. PSYCHOLOGICAL ASSUMPTIONS

The experiment to be reported involves determining chess positions that occur in normal play (tournament games) by asking questions. The conclusions drawn from this experiment rest on a number of psychological assumptions. These have been discussed in the literature, and are briefly reviewed here.

†Present address: Informatik, ETH, CH-8092 Zurich, Switzerland.

2.1. Assumption 1: Short-term and long-term memory, and chunks

People have a vast long-term memory. Reading from this memory is fast (seconds), but writing into it is a slow process that takes at least minutes, perhaps hours or days. Memorization requires repeated effort (writing into memory). People also have a tiny short-term memory, with instantaneous read/write access. The capacity of these two memories is measured in 'chunks' (G. A. Miller 1956). Long-term memory has a practically infinite capacity, but the capacity of short-term memory is 'the magic number 7 ± 2' chunks. A chunk is an ill-defined unit of storage that gets moved between short- and long-term memory. To most mathematicians, the number 3.14159 is one chunk, something that they recall and recognize at a glance. In a sequence of random digits, on the other hand, each digit recalled is a separate chunk.

2.2. Assumption 2: Recognition of chess positions by means of patterns

De Groot [1965] reports on experiments where subjects having different levels of chess skill were shown chess positions for a few seconds, then were asked to reconstruct the position seen. His main result can be summarized as follows. On realistic chess positions, taken from actual games, experienced players performed significantly better than novices. On random, but legal, chess positions, novices recalled positions as well as experienced players. The explanation for this phenomenon is the following. A player builds up a library of chess 'patterns' (configurations of pieces) that he sees frequently on the board, and stores it in his long-term memory. When presented with a new position, he codes it in terms of a few appropriate patterns from his library. If he is able to recall a position after seeing it for less than a minute, this means that he has stored the position entirely in his short-term memory; there was no time to memorize it in long-term memory. According to Assumption 1, he must have coded the position in no more than about seven chunks, or patterns. If a player can recall any realistic chess position after a short exposure, this means that his library of patterns is sufficiently large so that some combination of about seven patterns exists to match *any* position.

2.3. Assumption 3: No extraneous patterns

A player will not store in his long-term memory any patterns that are useless to him, that he has not seen before, and that do not occur in realistic positions.

3. THE NUMBER OF PATTERNS STORED IN LONG-TERM MEMORY AS A MEASURE OF CHESS-SPECIFIC KNOWLEDGE

De Groot's experiments suggest that one reasonable measure of a

persons's chess knowledge is the number of patterns he has committed to long-term memory for encoding chess positions, since this number appears to grow with the competence of the player. We are now interested in the maximum number of patterns that any player may have stored in long-term memory. We maintain that a player has achieved this maximum number if he can recall any realistic chess position after an exposure shorter than what would be needed to commit this position to long-term memory. Experience indicates that this ability is reached by players of intermediate strength—perhaps at the level of experts; in other words, that the number of patterns stored may increase rapidly at the lower levels of chess competence, but then reaches a saturation point. The experiments reported here were all performed with subjects who have the ability to look at an arbitrary realistic chess position for less than a minute, then recall it to perfection. According to our assumptions 1 and 2, this means that they have the maximal number of patterns. According to assumption 3, these subjects have exactly the number of patterns required to code any realistic chess position as a configuration of at most s patterns, where s is the number of chunks they can hold in short-term memory.

4. THE ENTROPY OF THE SPACE OF CHESS POSITIONS, AND HOW TO MEASURE IT

In the preceding paragraph, the number of patterns committed to long-term memory was related in a vague way to the number of 'realistic' chess patterns, i.e. those that have occurred, or might occur, in actual games. Since the vast majority of these 'realistic' positions have never occurred over the board (there are too many of them), and since positions that look 'unrealistic' do occur once in a while in actual play, there is a problem of how to count the number of realistic positions.

The well-known concept of *entropy* from statistical communication theory provides the needed measure. We say that any legal chess position might occur in actual play, but there is a probability of occurrence attached to each position. This probability is higher for realistic positions than it is for randomly constructed legal positions. Let $E = (e_1, \ldots, e_n)$ be the set of all legal chess positions, and let p_i be the probability of occurrence of position e_i. The entropy $H(E)$ of the space E, measured in bits, is as defined

$$H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i$$

where log is the logarithm to the base 2. Owing to the fact that the

positions that have actually occurred form such a vanishingly small sample from the space E of all positions, and that the majority of the positions that have occurred at all, have occurred only once, the probabilities p_i cannot be determined experimentally. The entropy $H(E)$, on the other hand, can be determined with reasonable accuracy by a version of the 'twenty-questions-game', as follows. A position that has occurred in actual play is chosen at random; a 'guesser' who does not know this position is asked to determine it by asking multiple-choice questions (including binary true/false questions). These are answered by an informant who does know the position. The validity of this questions-game for determining the entropy of a sample space is due to the 'Noiseless Coding Theorem' of communication theory. In a loose terminology which is sufficient for our purpose it states:

'Given a sample space E with entropy $H(E)$, and given that by sampling from E, a particular event e_i has occurred; on average, over many samples:

(1) it is impossible to determine which event e_i has occurred in less than H binary (Yes-No) questions

(2) it is possible to come as close as desired to H binary questions by asking the right questions'.

Asking the right questions, and getting correct answers to them, of course requires that the guesser and the informant bring their chess knowledge into play. Thus the approximate value of the entropy obtained by the questions-game will depend to some extent on the chess competence of the guesser and the informant. Since the mere recognition of positions is a rather low-level chess skill, it is plausible to assume that chess players who have reached a certain level of competence (for example, are able to play blindfold chess), have the required skill and terminology to ask and to answer 'the right questions'.

5. QUANTITATIVE RELATION BETWEEN THE NUMBER OF PATTERNS STORED IN LONG-TERM MEMORY, AND THE ENTROPY OF THE SPACE OF POSITIONS

Let c be the number of patterns a player has stored as chunks in his long-term memory. Let s be the number of chunks he can hold in his short-term memory. Then a player can code about 'c choose s', i.e.

$$\binom{c}{s} = \frac{c!}{s!(c-s)!} \approx \frac{c^s}{s!}$$

different positions. Under the various assumptions stated earlier, this

number is approximately equal to 2^H, the number of realistic chess positions. This leads to the approximation.

$$c \approx (s!2^H)^{1/s}$$

For reasonable values of s, i.e., the magic number 7 ± 2, and for various values of the entropy H, c assumes the approximate values shown below:

$s\backslash H$	50	60	70	80	90	100
6	750	3000	7500	30000	95000	300000
7	500	1500	3400	10000	27000	70000
8	250	1000	1800	3800	8000	24000

Unfortunately, the function $c(s,H)$ is very sensitive to both of its parameters, and thus a determination of the value of c by this method is inherently unreliable.

6. THE GUESSING GAME

When faced with an unknown position, the guesser is allowed any question that is pertinent to chess positions, and has a multiple-choice answer. (Binary yes/no answers are a special case of multiple choice.) Questions are *not* allowed that pertain to the game in which this position arose. Questions need *not* have an objective answer; the answer may depend on the subjective judgement of the informant. A few examples will make these conditions clear.

Not allowed: 'If White has an advantage, describe wherein this advantage consists' because it is not a multiple-choice question.

On the other hand if the guesser has already determined that White has an advantage, the following *is*

Allowed: 'Is White's advantage primarily due to (a) material superiority, or (b) positional superiority'. Notice how the informant's chess judgement enters into his answer.

Not allowed: 'Did White castle in this game?'

Allowed: 'Judging from the current position, would you say that (a) White has castled, (b) White has not castled, (c) you can't tell?'

Not allowed: 'Did White win this game?'

Allowed: Would you say White has a won position, Yes or No?'

Allowed: Is it an opening, middle game, or endgame position?'

Even if the informant has difficulty classifying the given position as opening or middle game, he has to make up his mind, and is not allowed to answer the last question by saying 'It could be called either an opening or a middle game position'. The latter, however, would be a fair answer to the five-way question 'Opening, between opening and middle game, middle game, between middle game and endgame, or endgame?

A question with k possible answers is counted as giving log k bits of information. For example, the question 'Tell me where the White King is' is allowed, and charged with log $64 = 6$ bits. Most questions actually asked have only a few choices. For your convenience if you wish to play the guessing game, here is the amount of information provided by a k-choice question for small values of k:

k	2	3	4	5	6	7	8
log k	1	1.58	2	2.32	2.58	2.81	3

7. THE EXPERIMENT

Our experiment to determine the entropy of the space of chess positions is still continuing. Positions are selected randomly from master-level tournament games. Guessers and informants are experts or masters. Guessing one position takes between half an hour and one hour. So far each of 10 positions has been guessed by two people. The average number of bits required to guess a position was about 70. The range was 50 to 80 bits, with the exception of one very wild position, which required 91 bits for one guesser, and 100 bits by another. Opening positions require fewer bits than middle game positions or endgame positions. The latter two types require about the same number of bits, which can be explained as follows: while an endgame position has fewer pieces, these can be in many more different places than they can in a middle game. For example, a King in an endgame can be anywhere on the board, while it is almost always restricted to a few squares in the middle game.

For comparison's sake: 136 bits suffice to determine *any* legal chess position (of which there are about 10^{40}). Simon and Gilmartin (1973) have estimated that a chess master may have stored 30,000 chunks of chess-specific knowledge†. Our experiments do not contradict this, but they are too weak to confirm it.

Acknowledgements

I am grateful to Donald Michie for his continuing stimulation of my interests in computer chess and the cognitive aspects of chess, and for his help with the preparation of this note. This paper is reprinted from *SIGART Newsletter*, **62**, 13–15, 1977, by kind permission of author and publisher.

†Chinese has about 50,000 ideograms. D.M.

REFERENCES

Chase, W. G. and Simon, H. A. (1973). Perception in chess. *Cognitive Psychology* **4**, pp. 55–81.

Groot, A. de (1965). *Thought and choice in chess*, (ed. G. W. Baylor). The Hague and Paris: Mouton. (Translation, with additions, of Dutch version of 1946.)

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychol. Rev.* **63**, pp. 81–97.

Simon, H. A. and Gilmartin, K. (1973). *Cognitive Psychology* **5**, pp. 29–46.

19

PROMIS: Experiments in Machine Learning and Protein Folding

R. D. King†
The Turing Institute
Glasgow, UK

Abstract

The aim of these experiments is to test the use of machine learning as a tool for forming theories from data. A machine-learning program (PROMIS) was developed to form rules for predicting protein secondary structure from primary structure—an important unsolved problem in molecular biology. PROMIS uses a top-down controlled hill-climbing beam search with the rules for predicting secondary structure being search states. Structured background knowledge is used to transform the search space and control generalization. Six rules were found that are humanly comprehensible and provide a chemically meaningful description of the important factors in formation of secondary structure. These rules predicted protein secondary structure with a Q_3 accuracy of 57 per cent, which is comparable with the most commonly used prediction methods. Variations of the rules were found with different accuracies. These were found to help highlight the important features of the rule. Rules were also found which used threshold logic to match sequences of primary structure. These rules were found to be suitable for predicting turn secondary structure. PROMIS is an example of the application of machine learning to molecular biological databases where there is an increasing demand for some form of automated discovery.

1. INTRODUCTION

Perhaps the most promising and yet most difficult application of machine learning is in the area of scientific discovery: 'the most technically gripping challenge, ... will be how to spread the computer wave from the front end of the scientific process, the telescopes, microscopes, ... spark chambers, and the like, back to recognition and reasoning processes by which the chaos of data is finally consolidated into orderly discovery' (Michie 1982). For scientific discovery, machine learning is viewed as a tool to aid working scientists in forming theories from data. Such tools

†Present address: Brainware, Gustav-Meyer-Allee 25, 1000 Berlin 65, FRG

are needed because it often proves difficult for a scientist to perceive patterns in data, even though strong patterns exist. Difficulty in perceiving patterns may occur for a number of reasons: for example patterns may be obscured because there is a very large amount of data, or because the data may be in a difficult form. This paper describes PROMIS (protein machine induction system), a program designed to aid in the formation and creation of theories about the formation of protein secondary structure from primary structure (King 1987).

1.1. The problem

Proteins are the most complicated chemicals that exist. They are responsible for almost all the important tasks within living systems. In a human it is estimated that there are around 100,000 different types of protein. Typical types of proteins are: enzymes (e.g. DNA polymerase), transport proteins (e.g. haemoglobin), protection proteins (e.g. HIV antibody), and toxins (e.g. cobra venom). The conformation of a protein, its three-dimensional shape, determines its function. Anfinsen *et al.* (1961), in an elegant series of experiments, showed that for a given environment, the conformation of a protein is uniquely coded for in the one-dimensional structure of a gene. It is now possible in molecular biology to create a gene and to have this gene translated into the three-dimensional shape of a protein: yet it is not possible to create new useful proteins by doing this, as the rules governing the conversion of the one-dimensional information into three-dimensional information and the rules relating conformation to function are not understood. PROMIS is concerned with finding rules relating one-dimensional and three-dimensional information—'the protein folding problem'.

A gene forms the conformation of a protein in the following way: the DNA sequence is first translated by use of the genetic code into a sequence of amino acids which is the primary structure of a protein (via a matching RNA sequence), the one-dimensional primary structure then spontaneously folds itself into the final conformation of the protein. Between the primary structure and the conformation there is a level of structure known as the secondary structure (Schulz and Schirmer 1978). In finding rules relating the primary structure and the conformation, it is simplest to split the problem into rules relating primary to secondary structure and rules relating secondary structure to conformation (Cohen *et al.* 1982; Lathrop *et al.* 1987). PROMIS is designed to discover rules that convert primary structure into secondary structure. The difficulty of predicting secondary structure comes from the fact that it is the sequence of primary structure as a whole which determines the secondary structure of any particular position, and so any individual amino acids's contribution cannot be said to be context free. In PROMIS, this

Godian knot is cut by considering long-range interactions to be 'noise' and local regions of secondary structure to be caused by the corresponding local region of primary structure, an assumption of locality.

It is thought probable that much of the knowledge necessary for predicting protein secondary structure already implicitly exists in data bases, hidden by the bulk and difficult nature of the information. There are around 70 proteins of known primary and secondary structure, which together give around 10,000 positions of primary structure where the corresponding secondary is known. The secondary structure information has been acquired by protein crystallography and is very difficult to obtain. The primary structure information is much more easily obtained by genetic sequencing methods. In recent years there has been an explosive growth in genetic sequence information and there are now around 10,000,000 primary structure positions known (Smith 1987). The great imbalance of information is set to get even worse with the prospective sequencing of the human genome (Roberts 1987). A solution to the protein folding problem would allow us to exploit this increase in information fully by removing the bottleneck of crystallography.

1.2. The suitability of the problem

The problem of predicting a protein's secondary structure from its primary structure is increasingly becoming a test bed for applications of machine learning. There are several reasons for this:

1. It is of the highest scientific importance.
2. It is of potentially great practical importance.
3. It is a well-known hard and intractable subject and as such presents a great challenge to machine learning technology.
4. Human and statistical methods have fared poorly in attempting to find regularities in the data and solve the problem.
5. There exists a large and growing amount of symbolic data of relevance to the problem (consisting of example proteins of known primary and secondary structure).
6. There exists relevant background knowledge in a form that can readily be applied in a machine-learning program.
7. The data is available in a machine-readable form.
8. There is a reasonably well accepted measure of success, allowing comparison between different machine learning techniques and also more conventional methods.

1.3. Previous work

There have been three types of traditional approach to the problem of secondary structure prediction: methods based on statistics, methods based on chemical theory, and most recently methods based on homology (exemplars), (Sternberg 1983). The most successful achieve an accuracy of around 60 per cent. Statistical methods examine the data base of known primary and secondary structures to find statistical trends, little domain knowledge is used and the rules produced are not in a form comprehensible by people or related to chemical theory, e.g. Gibrat *et al.* (1987). Chemical theory methods use knowledge of molecular structure to produce prediction rules. These rules are comprehensible but mainly ignore the empirical evidence of the data base of known protein structures, e.g. Lim (1974a, b) and Cohen *et al.* (1983). The homology-based methods use domain knowledge to match unknown sequences with known sequences to make their predictions. These predictions suffer by having no explanation in chemical theory and by not producing any new knowledge; the methods closely resemble exemplar-based learning algorithms, e.g. Levin *et al.* (1986).

Apart from PROMIS, two other machine learning approaches have been applied to protein secondary structure prediction. Qian and Sejnowski (1988) made an extensive study of the application of neural networks to the problem. This work achieved an impressive accuracy of prediction and raised several important points about protein secondary structure prediction, but it had the disadvantages of involving a large number of numerical parameters and treating protein folding as a black box with no explanation in human comprehensible terms (a Hinton diagram means nothing to a molecular biologist). Seshu *et al.* (1988) applied the learning program PLS1 to the problem (this program is similar to ID3). PLS1 failed to achieve results significantly better than the default accuracy. They also applied their program NTC (New Term Constructor) to the problem. NTC consists of a complex suite of programs designed to carry out constructive induction in 'hard' domains (Rendell 1988). NTC achieved reasonable results but did not produce concepts comprehensible to molecular biologists.

2. METHODS

The learning problem is: given the proteins of known primary and secondary structure, find generalized relationships between the existing primary and secondary structure which can be used to predict an unknown secondary structure from a known primary structure. Inductive learning is taken to be a heuristic search for a goal through a space of symbolic descriptions generated by application of various rules of inference to the initial observational statements (Mitchell 1982). The

search method used was designed specifically for induction of strings in the presence of noise over a large search space.

2.1. Inputs

The primary and secondary structure of a protein can be considered to be two related strings of characters, where there is a one-to-one mapping between the primary structure and the secondary structure.

[$p1$, $p2$, $p3$, $p4$, $p5$, ..., pn] primary structure [] brackets indicate a sequence.
[$s1$, $s2$, $s3$, $s4$, $s5$, ..., sn] secondary structure

In the alphabet of life there are 20 letters in the primary structure and three letters in the secondary structure. The primary letters are represented as follows (p, g, c, a, s, n, v, t, d, i, l, m, f, y, w, h, k, e, r, q); these are the 20 genetically coded amino acid residues; () brackets indicate a set. The secondary lettes are (A, B, T). These are the three types of secondary structure, alpha-helix, beta-sheet, and turns, respectively.

2.2. Outputs

The concepts induced by the learning program should be good descriptions of the data and useful in prediction. The concepts should also be comprehensible to the domain scientists using the program, that is, they should fit in with current scientific ideas about the domain and perhaps even the scientist's own biases. The concepts should also be simple enough to be represented in a machine learning program.

The output of PROMIS is rules that predict secondary structure from primary structure. The general form of the rules is:

if the string of classes [CC, Dc, Ec] occurs
(i.e. a string of residues occurs [w, x, y]
where the residue w belongs to the class Cc
where the residue x belongs to the class Dc
where the residue y belongs to the class Ec)
then the residues are all in the secondary stucture type S.

For example, using the rule [positive, negative, positive] → A

with the positive class = (h, k, r)
with the negative class = (d, e)

then the primary sequence

[h, d, r] is predicted to have
[A, A, A,] as a corresponding secondary structure by use of the rule.

These rules are similar in form to that used by domain experts in encoding knowledge about proteins.

Learning is carried out in a representation that is different from the input data: that is, not at the residue level (Barr and Feigenbaum 1983, Dietterich and Michalski 1983). Background knowledge is used to group the residues into classes sharing a particular chemical property, or conjunction or disjunction of several properties. For example the residues (*r, k, h*) form the class of 'positive' residues and the residues (*d, e*) form the class of 'negative' residues. There is also the class 'charged' which consists of the residues (*r, k, h, d, e*) and is the conjunction of the classes 'positive' and 'negative'. The reason for grouping residues into classes is to be able to produce rules that can specify more than one primary sequence (that is, they are more general). There are 71 classes used and each residue is a member of 30 classes on average, which means that for a primary sequence of length n there are around 30^n possible class sequences. The classifications used in this work are those of Taylor (1986).

2.3. Background knowledge used as search operators

The class representation of the residues is equivalent to a generalization hierarchy. The graph is a directed acyclic graph and not a tree because any particular node, with the exception of the root node and its children, can have more than one parent; this is because a particular class can be a subclass of several different complex classes; it is a tangled hierarchy. The root node in this example is the class of all residues ('all'). Generalization structures are described by Michalski and Stepp (1983).

The transformation of the set representation into a generalization lattice immediately suggests the method of induction known as climbing a generalization tree (Michalski 1983). This is based on the fact that the ancestor nodes of a set consist of inductive generalizations of that set. Thus, to carry out induction a rule containing a set can be generalized to a rule containing a set that is an ancestor of the original set.

This can be more formally represented thus:

if the rule exists $[Bs] \rightarrow A$,

where *Bs* and *Cs* are sets, *Cs* is an ancestor of *Bs* and *A* is a secondary conformation type, then the following inductive inference can be made:

$[Cs] \rightarrow A$.

A possible example of this is:

from [positive] $\rightarrow A$, infer the generalization [charged] $\rightarrow A$.

Because of the nature of the complex classes some of the climbing tree inductions can be represented as dropping conditions or adding alternatives. An example of dropping a condition is the generalization of [large_and_polar] to [large]. In this example the condition of polarity

has been dropped as the set [large] is an ancestor of [large_and_polar]. An example of adding an alternative is the generalization of [small] to [small_or_polar]. In this example the alternative of polarity has been added as the set [small_or_polar] is an ancestor of [small].

Specialization can easily be carried out with the use of the generalization tree by simply reversing specialization and moving down the tree. A possible example of this is:

from [charged] → A, infer the generalization [positive] → A.

A method of increasing the length of the string is also needed. This can be achieved by adding a new class to either end of the string of classes, a form of specialization specific to string induction. A possible example of this is:

[positive] → A, becomes [positive, negative] → A

The operators used in PROMIS were restricted to: lengthening one end of a rule at a time by adding a new class to the rule's condition and using the generalization tree operators to generalize and specialize on one class of the rule's condition at any one time.

2.4. Rule evaluation

The goal of the search is to find general powerful rules for converting primary structure into secondary structure. To do this a rule evaluation function and a method of assessing statistical significance are needed. The existence of a very large amount of noise in the data (associated with the restrictions inherent in our data representation) means that the best rules should not be expected to be necessarily 100 per cent accurate. It is also expected that no single rule will have 100 per cent coverage.

To find the evaluation of a rule, the sections of primary and secondary sequence are collected where the rule applies in the data base. The sequences of actual secondary structure are then compared with the predicted secondary structure to count how many positions were correctly predicted and how many positions were incorrectly predicted. The evaluation function used is:

$$(P - N)/(P + N + M);$$

where P = the number of correctly predicted positions, N = the number of incorrectly predicted positions and M = the number of positions not predicted. For example: if the primary sequence

[f, g, h, h, g, h]

is found to have the following secondary structure in the data base

[A, A, A, B, B, B]

and it is predicted to have the following secondary structure by a rule

$[A, A, A, A, X, X]$ (X = no prediction made)

then the number of correctly predicted positions is three, the number of incorrectly predicted positions is one, the number of positions not predicted is two and the evaluation of the rule is 0.333. The justification of this evaluation function is that it increases with correct predictions, decreases with incorrect predictions, and is normalized for a given example set; a similar evaluation functions is used in NEWGEM (Mozetic 1986).

It is important that any relationship between primary and secondary structure that is found in the data base should be statistically significant. This is because the rule is to be used to predict unknown secondary structure in the future, and thus must represent a real relationship, not just one that arises through the chance existence of particular primary and secondary structures within the data base. As a heuristic for finding significant rules, a threshold test is used in PROMIS. This involves introducing a threshold number of positions which a rule must cover before it is considered to be significant. For example, if the threshold is set at 100, then the number of correctly predicted and incorrectly predicted positions must be > 100. A similar method is used in RULEGEN from Meta-DENDRAL (Buchanan and Feigenbaum 1981), in SEQUOIA (Haiech *et al.* 1986—SEQUOIA also includes a threshold for incorrect coverage) and in the work of Rooman and Wodak (1988).

2.5. Control of search

Complete search of the rule space is not possible. It is therefore necessary to use some form of heuristic search. The method adopted uses top-down 'generate and test' control, because additional heuristics can be easily applied and because it has good noise immunity (Mitchell 1982). This method has the disadvantage that it involves many passes through the data.

This algorithm (see below) is a form of hill-climbing beam search. It was chosen as it avoids the large memory requirements of best-first search while still avoiding premature commitment to a particular branch of the search tree, see Bisiani (1987). Beam search is used in many induction programs, as, for example, AQ15(Michalski *et al.* 1986) and CN2 (Clark and Niblett 1987).

Algorithm

```
begin
  add an initial beam set of rules
  repeat
    new rules are generated from the beam set by use of the operators,
```

the new rules are evaluated,
store any rule from the beam set that does not produce a better rule,
the beam set becomes the best evaluated new rules,
until
the beam set is empty or a set number of iterations have passed without a rule entering the store with the highest known evaluation,
the best rule found is that with the highest evaluation in the store.
end

The examples of the best rule may then be covered and the process repeated.

In the search PROMIS faces the difficult problem of knowing when to stop and accept a local maximum as the best that can be found, given the limited resources of time and space allowed for the search. Many, if not most practical search problems suffer from lack of a simple test to tell when the goal state has been found. This limits the value of the traditional search formalism and algorithms that exploit it. The method of allowing several iterations to occur between finding the best rule and stopping, is a compromise between best-first and hill-climbing search and allows some local maxima to be avoided.

3. RESULTS

3.1. Data

The example set of proteins used came from the standard Brookhaven data base, via the molecular biology laboratory of Birkbeck College in the University of London. The secondary structure is objectively designated using a modified algorithm from Kabsch and Sander (1983a), which assigns secondary structure on the basis of a known tertiary structure. The proteins used were selected from the data base by M Sternberg, an expert in the subject of protein structure. The Brookhaven data base contains ~ 100 proteins. This was trimmed down to 61 proteins by removing polypeptides and homologous proteins. In addition, only one polypeptide chain was selected from any protein; this was done to make the data as unbiased as possible. The test and training set were split randomly to give a 7:3 division. There are 8024 positions in the training set; 2161 are alpha-helices, 1466 are beta-sheets, and 4397 are turns. There are 3283 positions in the test set, 917 are alpha-helices, 668 are beta sheets and 1698 are turns.

3.2. Experiment 1

PROMIS was used to find general rules for predicting secondary structure from primary structure and six rules were found (King 1988) (see Table 1). All the rules were found starting with the rule,

Table 1. Individual evaluation of rules found. Rules 1a, 2a and 3a are for predicting alpha-helices, rules 1b and 2b are for predicting beta-sheets, rule 1t is for predicting turns. 'Evaluation' is the evaluation function described above in section 2 *Methods*. '% covered' is the amount of secondary structure of the type predicted covered. '% correct' is the accuracy of the prediction for the positions covered. '% correct based on frequency of secondary structure' gives the frequency of the predicted type of secondary structure in the test set. The decrease in coverage and accuracy from the training set to the test set probably represents some overfitting of the rules on the training set.

	On training data			On test data			% correct based on frequency of secondary structure
Rule	Evaluation	% covered	% correct	Evaluation	% covered	% correct	
1a	0.0339	17	79	0.0079	13	56	28
2a	0.0147	12	64	−0.0006	8	49	28
3a	0.0242	12	78	0.0155	7	83	28
1b	0.0067	16	57	0.0049	17	54	20
2b	0.0014	10	52	−0.0006	3	48	20
1t	0.1639	78	62	0.1167	81	58	52

[all] → required secondary structure;

where 'all' matches every type of primary structure and the required secondary structure is either alpha-helix, beta-sheet, or turn.

The beam size was 10, made up as follows: first the highest-scoring rule according to the evaluation function; then three rules selected to be different in at least two places from this rule; finally the next highest-scoring six rules, making 10 in all.

These rules were found to perform comparably with the best published claims for rules produced by domain experts. Exact comparison is difficult because of the imprecise reports of the hand-produced rules, see, for example Cohen *et al.* (1983).

All the rules found show agreement with accepted knowledge about the chemistry and structure of proteins. Yet the only knowledge about proteins that was coded into PROMIS was about residue classes. The higher-level features found in the rules such as the amphilicity in alpha-helix rules and hydrophobic cores in beta-sheet rules were found empirically by PROMIS. As little chemical knowledge was built into PROMIS, discovery of these concepts gives credence to the rules. The lack of knowledge also allows PROMIS not to be bound to existing theory and lets it form new and potentially useful concepts about proteins. Some such concepts have been found and are being further investigated in collaboration with domain experts.

The typical form of the rules is illustrated by the first rule to be found for predicting alpha-helix secondary structure (rule 1a) (see Figure 1 and Table 2).

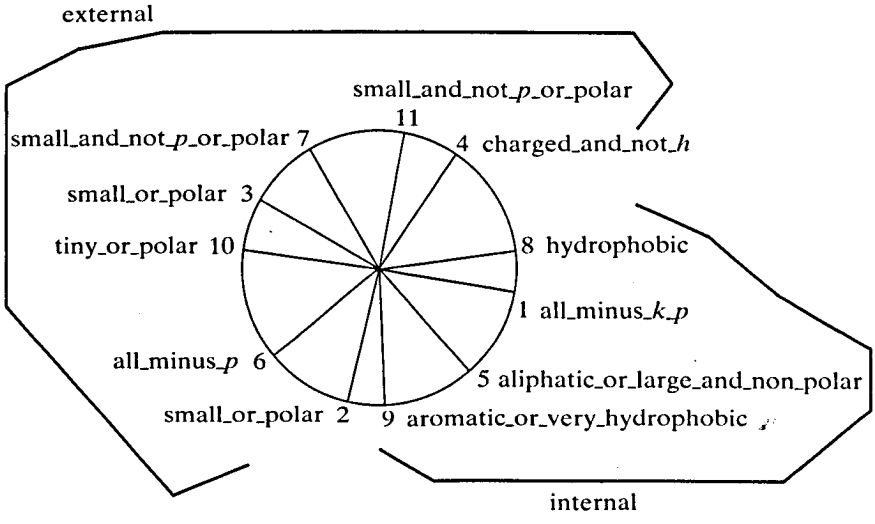


Figure 1. Helical wheel plan of rule 1a. This shows the actual position of the classes in an alpha helix.

Table 2. The occurrences of rule 1a in the test set. The 'protein id' is the short standard identification name of the protein; 'pos' is the sequence position of the first amino-acid residue covered by the rule; 'secondary structure' is the sequence of secondary structure occurring (the rule predicts the sequence to be all 'a' alpha-helix type), 'primary structure' is the sequence of secondary structure from which the rule predicts the primary structure.

protein id	pos	secondary structure	primary_structure
1ABP	19	[a a a a a a a a a a a a]	[t e w k f a d k a g k]
1SBT	38	[t t t t t t t t t b b b]	[s h p d l k v a g g a]
1TIM	1	[t t t t t b b b b b t]	[a p r k f f v g g n w]
1TIM	50	[a a a a t t t t t b b b]	[a r q k l d a k i g v]
1TIM	54	[t t t t b b b b b b t]	[l d a k i g v a a q n]
1TIM	108	[a a a a a a a a a a a t]	[i g q k v a h a l a e]
1TIM	179	[a a a a a a a a a a a a]	[q a q e v h e k l r g]
1TIM	183	[a a a a a a a a a a a a]	[v h e k l r g w l k t]
1TIM	201	[a a t b b b b t t t t]	[v q s r i i y g g s v]
1TIM	235	[t t t a a a a a a t t]	[l k p e f v d i i n a]
2ADK	24	[a a a a a a a a a t t t]	[q c e k i v q k y g y]
2ADK	101	[a a a a a a a a a t t t]	[q g e e f e r k i g q]
2ADK	146	[a a a a a a a a a a a a]	[i k k r l e t y y k a]
2CNA	57	[t t t b b b b b b b t]	[v d k r l s a v v s y]
2MDH	299	[a a a a a a a a a a a a]	[s r e k m n e t a k e]
351C	26	[a a a a a a a a a a a t]	[a y k d v a a k f a g]
3DFR	34	[t t t t b b b b b a a]	[t v g k i m v v g r r]
3DFR	75	[b b t t a a a a a a a]	[v v h d v a a v f a y]
4FXN	11	[a a a a a a a a a a a a]	[n t e k m a e l i a k]
8PAP	153	[t t t t t t b b b b b]	[c g n k v d h a v a a]

[all_minus_*k_p*, small_or_polar, small_or_polar,
charge_and _not_*h*, aliphatic_or_large_and_non_polar,
all_minus_*p*, small_and_not_*p*_or_polar, hydrophobic,
aromatic_or_very_hydrophobic, tiny_or_polar,
small_and_not_*p*_or_polar] → Alpha-helix

In Figure 1 the most important thing to note is the amphiphilic nature of the rule, that is, the hydrophilic and hydrophobic residues are positioned on separate sides of the helix, corresponding to the faces of the helix which face externally (hydrophobic) and internally (hydrophilic); this separation is thought to be a major explanation for helix formation. The hydrophobic residues are arranged in the classic sequence $n(5)$, $n+3$ (8), $n+4$ (9); the same is true for the hydrophilic

residues $n(7)$, $n+3$ (10), $n+4$ (11) and $n-4$ (3), $n-3$ (4), $n(7)$, (Schiffer and Edmundson 1967).

The six prediction rules found are put together to produce a complete method for predicting protein secondary structure from primary structure. This method takes as an input a sequence of primary structure and produces as an output the corresponding secondary structure. Such a prediction method is directly comparable with any other secondary structure prediction method. The six rules combined to produce a Q_3 value of 63 per cent in the training data and 57 per cent in the test data. This accuracy is comparable with the most commonly used prediction methods such as that of Chou-Fasman, Robson, and Lim (Kabsch and Sander 1983b). When combined with protein domain-type specific rules obtained in the same machine learning study (King 1988), they produced a Q_3 value of 67 per cent for secondary structure prediction in the training data, and 60 per cent in the test set. This accuracy is comparable with the best available other methods for protein prediction.

3.3. Experiment 2 (variations on a theme)

In experiments with an alternative rule evaluation method, a threshold percentage accuracy *Ta* is set. If a rule meets this accuracy the number of positive examples is then maximized while still maintaining the heuristic for rule significance. The starting place of the search in these experiments has been the rules generated in Experiment 1.

The aim of these experiments is to find variations of successful rules which have different accuracies. This, it is hoped will highlight the important constant features of a rule across the range of accuracies. For example: if a successful rule is found with an accuracy of 65 per cent, then variations of the rule might be sought with an accuracy of >80 per cent; the resulting rule is likely to have fewer examples, but it will show what features are important in making the rule more accurate; conversely if a lower accuracy is set then the most general features of the rule will tend to show through and possibly make the rule more comprehensible to a domain expert.

The results of searching for rules of varied accuracy are illustrated by the case of rule 1a, the first rule found for predicting alpha helices (Figure 2). The highest accuracy rule was found by the specialization of only three classes. The rule selected for with an accuracy of >80 per cent was found by extending the length of the rule (a form of specialization). The class added was 'hydrophobic_or_small' which fits in with the amphiphilicity of the rule. Lower accuracy rules were found by generalizing the hydrophobic classes and extending the rule with the class 'all'. Interestingly the position of the new classes 'all' is such that they lie between the external and internal faces of the helix. There is an

Selection	Highest	>80%	Best	>70%	>60%	>50%	>40%
Coverage	10%	16%	17%	22%	26%	33%	69%
1	←	←	all_minus_k_p	→	→	→	*all*
2	←	*small_or_polar_ and_not_ aromatic*	small_or_polar	→	→	→	*all*
3	←	←	small_or_polar	→	→	→	→
4	←	←	charged_and_ not_h	→	→	*charged*	*all_minus_p*
5	←	←	aliphatic_or_ large_and_non_ polar	→	*aromatic_or_ very_ hydrophobic*	*hydrophobic*	*aromatic_or_ very_ hydrophobic*
6	←	←	all_minus_p	→	→	→	→
7	*small_and_not_ p_or_ hydrophilic*	←	small_and_not_ p_or_polar	→	→	→	→
8	←	←	hydrophobic	*hydrophobic_ or_small_and_ not_p*	→	*hydrophobic_ or_small_and_ not_p*	→
9	*aromatic_or_ aliphatic_or_m*	←	aromatic_or_ very_ hydrophobic	→	→	→	→
10	*tiny_or_polar_ and_not_ aromatic*	←	tiny_or_polar	→	→	→	→
11	←	←	small_and_not_ p_or_polar	→	→	→	→
+1		*hyrophobic_or_ small*		*all*	*all_minus_p*		
+2				*all_minus_p*			

Figure 2. Variations of rule 1a selected for at different 'Selection' accuracies. The 'Coverage' is the amount of coverage obtained at this accuracy. The 'highest' accuracy is the highest possible accuracy that still allowed the threshold number of examples to be found. The 'best' accuracy is rule 1a. The order of classes goes from top to bottom. An arrow through a box means that it contains the same class as 'best' for that position, a cross means that no class exists.

inversely proportional relationship between correctness of prediction and coverage; the higher the accuracy the lower the coverage (Table 3). Both coverage and correctness decrease in the test set although coverage decreases less. The only rule that does not vary much between training and test is the lowest accuracy rule. This rule manages to cover 60 per cent of all alpha-helix residues with an accuracy that is too low for direct use in prediction but still much higher than the percentage of alpha-helix in the data base (28 per cent).

Table 3. The accuracy, coverage, and evaluation of variations of rule 1a selected for at different accuracies with a constant threshold; 'high' is the highest possible accuracy found for the threshold number of examples, 'best' is the original rule 1a.

	On training data			On test data		
Rule	Evaluation	% covered	% correct	Evaluation	% covered	% correct
high	0.0248	10	89	0.0057	9	70
>80	0.0332	16	81	0.0027	12	56
best	0.0339	17	79	0.0032	13	56
>70	0.0338	22	70	0.0021	14	47
>60	0.0237	26	60	0.0027	18	47
>50	0.0006	33	50	0.0142	23	39
>40	−0.0884	69	40	−0.0376	60	39

3.4. Experiment 3

This experiment is based on using a form of threshold logic in the representation of rules. The idea is that a rule can be said to match a primary structure even if all the class positions of the rule do not match the primary structure exactly. An example makes this clearer:
The rule

[positive, negative, positive] → A
positive = (h, k, r)
negative = (d, e)

makes a mistake in matching the primary sequence

[h, t, r]

(the mistake being that t is not a member of the set 'negative'), therefore the primary sequence [h, t, r] cannot be said to match the rule head [positive, negative, positive]. However, if one mistake in matching was allowed the sequence would be considered an example of the rule.

Allowing one error in matching a rule is equivalent to several rules, each rule having one position as a 'wild card'.

For example, allowing one error in matching the rule

[positive, negative, positive] → A,

is equivalent to the three rules:

[all, negative, positive] → A
[positive, all, positive] → A
[positive, negative, all] → A,

It is simpler to write down one rule and allow one error in matching, than to write down the variations of the rule with the class 'all'; this is especially so if the rule is of greater length.

It is hoped that, by changing the representation in this way, and allowing mistakes in matching, more powerful rules will be found. These should cover more examples of a particular secondary structure while still retaining high accuracy and most importantly human comprehensibility. Mistake matching implies a model of secondary structure where every position in a primary sequence is not vital in forming the secondary structure, and where any one position can be substituted without certain loss of the corresponding secondary structure.

The use of various degrees of matching is a common idea in pattern matching (Slagle and Gini 1987). Mistake matching is also related to the technique for dealing with noise known as 'flexible interpretation' of rules (Michalski *et al.* 1986, Michalski 1987). The idea also receives support from the fact that it is one of the types of representation that has already been used in protein structure prediction (Cohen *et al.* 1983, Cohen *et al.* 1986); however, no attempt has been made to evaluate the representation's suitability for the problem.

The usefulness of normal rules (with complete matching) was compared with rules that make one mistake in matching. For 13 different rule types and appropriate splits of the data, rules were sought for both strict matching and mistake matching and the results compared (King 1988). Mistake matching rules were found to be less successful than normal rules in describing and predicting alpha-helices and beta-sheets, but more successful in describing and predicting turns; the difference between the two types of rule is more marked in the test set than in the training set. The different success rate of mistake matching rules in describing and predicting secondary structure types is probably mainly due to the structural form of the different secondary types. In alpha-helices and beta-sheets, every position is important and the inclusion of an incompatible residue at a position may disrupt the whole protein structure, e.g. a hydrophilic residue within the internal face of an alpha-helix. In a turn, every position is not so vital and residues can be added without disrupting the whole structure of the protein, e.g. it is known that mutations in proteins tend to occur in turns (Thornton 1986). This difference in structural nature between turns and other types of secondary structure means that mistake matching rules are badly suited for describing alpha-helices and beta-sheets but well suited for describing turns.

The two types of rule often resembled each other, suggesting that they were just different ways of describing the same regularity in the data. For example:

The first rule to be found for predicting turns was:

[all, tiny_or_small_and_polar, all, tiny_or_polar_and_not_aromatic_or_*p*] → *T*

with an evaluation of 0.1167 in the test data. The corresponding mistake matching rule was:

[glycine, small_and_polar_or_*p*, all, tiny_or_polar_and_not_aromatic_or_*p*] → *T*

with an evaluation of 0.1648 in the test data.

4. DISCUSSION AND CONCLUSION

Molecular biologists are finding themselves submerged in information about macromolecular structure (von Heijne 1988). The amount of such information is already so great that it is beyond human ability to digest it all; and with the development of faster sequencing machines and the proposed sequencing of the human genome the amount of information is set to increase by several orders of magnitude.

The data are held in very large data bases, the most important of which are GenBank which contains DNA sequence data (Los Alamos National Laboratory, Los Alamos, New Mexico 87545, USA), PIR which contains protein primary structure data (National Biomedical; Research Foundation, Georgetown University Medical Center, 3900 Reservoir Rd., NW, Washington DC 20007, USA) and Brookhaven which contains tertiary structure (Brookhaven National Laboratory, Upton, New York 11973, USA). These data bases hold many important secrets, some of which should be capable of being discovered by machine learning. Possible problems in molecular biology to which PROMIS or other machine-learning programs could be applied are recognition of patterns for: antigen binding sites, prediction of RNA secondary/tertiary structure, protein initiation sequences in MRNA, protein coding sites in DNA, gene intron/extron juncture sites, DNA transcription promoters, etc. (Haiech *et al.* 1986).

PROMIS has learned rules that predict secondary structure with an accuracy comparable with other existing prediction methods. In contrast to most other competing methods, the rules are in a form that is humanly comprehensible and they represent theories about the formation of protein secondary structure. The true importance of these rules will only be discovered by a more general airing of the rules to workers in the subject.

It was found that variations in the generality of a rule could highlight its important features to molecular biologists and provide aids to their

understanding. A form of rule representation allowing mistakes in matching the conditional part of the rule was found to be unsuitable for predicting alpha-helices and beta-sheets but suitable for predicting turns.

Protein secondary prediction is a well-known difficult problem. It is therefore unreasonable to expect the problem to be solved easily by the application of any single new method. However, the inductive learning approach seems to be capable of making a useful contribution to solving the problem.

Acknowledgements

I would like to thank my supervisors Peter Mowforth and Profesor McGregor, along with my domain expert Mike Sternberg. I would also like to thank Professor Michie and Pete Clark for their advice. This work was supported by a grant from the Department of Computer Science at Strathclyde University and a grant from the SERC.

REFERENCES

Anfinsen, C. B., Harber, E., Sela, M., and White, F. H. (1961). The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. *Proceedings of the National Academy of Sciences (U.S.)* **47**, 1309–14.

Barr, A. and Feigenbaum, E. A. (eds) (1983). *The Handbook of Artificial Intelligence.* Pitman, London.

Bisiani, R. (1987). Beam Search. In *Encyclopaedia of artificial intelligence* (eds S. C. Shapiro, and D. Eckroth) pp. 56–57. Wiley Interscience.

Buchanan, B. G. and Feigenbaum, W. A. (1981). Dendral and Meta-Dendral: their application dimension. In *Readings in Artificial Intelligence* (eds B. L. Webster and N. J. Nilsson) pp. 313–22. Tioga, Palo Alto, Ca.

Clark, P. and Niblett, T. (1987). Induction in noisy domains. In *Progress in machine learning* (eds I. Bratko and N. L. Lavrac) pp. 11–30. Sigma Press, Wimslow, England.

Cohen, F. E., Sternberg, M. S. E., and Taylor, W. R. (1982). Analysis and prediction of the packing of alpha-helices against a beta-sheet in the tertiary structure of globular proteins. *J. Mol. Biol.*, **156**, pp. 821–62.

Cohen, F. E., Abarbanel, R. M., Kuntz, I. D., and Fletterick, R. J. (1983). Secondary structure assignment for alpha/beta proteins by a combinatorial approach. *Biochemistry*, **22**, pp. 4894–905.

Cohen, F. E., Abarbanel, R. M., Kuntz, I. D., and Fletterick, R. J. (1986). Turn prediction in proteins using a pattern-matching approach. *Biochemistry*, **25**, pp. 266–75.

Dietterich, T. G., and Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In *Machine learning: an artificial intelligence approach* (eds R. S. Michalski, J. Carbonell, J. G. and T. Mitchell) pp. 41–81. Tioga, Palo Alto, Ca.

Gibrat, J. F., Garnier, J., and Robson, B. (1987). Further development of protein secondary structure prediction using information theory: new parameters and consideration of residue pairs. *J. Mol. Biol.* **198**, pp. 425–43.

Haiech, J., Quinqueton, J., and Sallantin, J. (1986). SEQUOIA: Concept formation from sequential data. *Proc. EWSL-86*, Paris.

Kabsch, W. and Sander, C. (1983a). Dictionary of protein structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, **22**, pp. 2577–637.

Kabsch, W. and Sander, C. (1983b). How good are predictions of protein secondary structure? *F.E.B.S. Letters*, **155**, pp. 179–82.

King, R. D. (1987). An inductive learning approach to the problem of predicting a protein's secondary structure from its amino acid sequence. In *Progress in machine learning* (eds I. Bratko and N. L. Lavrac) pp. 230–50. Sigma Press, Wimslow, England.

King, R. D. (1988). A machine learning approach to the problem of predicting a protein's secondary structure from its primary structure. Ph.D. Thesis, University of Strathclyde, U.K.

Lathrop, R. H., Webster, T. A., and Smith, T. F. (1987). ARIADNE: Pattern-directed inference and hierarchical abstraction in protein structure recognition. *Communications of the ACM* **30**, pp. 909–21.

Levin, S. M., Robson, B., and Garnier, J. (1986). An algorithm for secondary structure determination in proteins based on sequence similarity. *F.E.B.S.* **205**, pp. 303–8.

Lim, V. I. (1974a). Structural principles of the globular organization of protein chains: a stereochemical theory of globular protein secondary structure. *J. Mol. Biol.* **80**, pp. 857–72.

Lim, V. I. (1974b). Algorithm for prediction of alpha-helical and beta-structural regions in globular proteins. *J. Mol. Biol.* **80**, pp. 873–94.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In *Machine learning: an artificial intelligence approach* (eds R. S. Michalski, J. Carbonell, J. G. and T. Mitchell) pp. 83–134. Tioga, Palo Alto, Ca.

Michalski, R. S. (1987). How to learn imprecise concepts: A method for employing a two tiered knowledge representation in learning. *Proc. Fourth International Workshop on Machine Learning*, pp. 50–8. Morgan Kaufmann, Los Altos, Ca.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proc. A.A.A.I.–5*, pp. 1041–5. Morgan Kaufmann, Los Altos, Ca.

Michalski, R. S., and Stepp, R. E. (1983). Learning from observation: conceptual clustering. In *Machine learning: an artificial intelligence approach* (eds R. S. Michalski, J. Carbonell, J. G. and T. Mitchell) pp. 331–64. Tioga, Palo Alto, Ca.

Michie, D. (1982). *Machine intelligence and related topics.* Gordon and Breach Science Publishers.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, **18**, pp. 203–26.

Mozetic, I. (1986). Knowledge extraction through learning from examples. In *Machine learning: a guide to current research* (eds T. Mitchell, J. Carbonell, and R. S. Michalski) pp. 227–31. Kluwer Academic Publishers.

Qian, H. and Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins through neural network models. *J. Mol. Biol.* **202**, pp. 865–84.

Rendell, L. (1988). Learning hard concepts. *Proc. International Workshop in Change of Representation and Inductive Bias–I.* pp. 70–100.

Roberts, L. (1987). New sequencers to take on the genome. *Science*, **238**, pp. 271–3.

Rooman, M. J. and Wodak, S. J. (1988). Identification of predictive sequence motifs limited by protein structure data base size. *Nature*, **335**, pp. 45–9.

Schiffer, M., and Edmundson, A. E. (1967). Use of helical wheels to represent the structures of proteins and to identify segments with helical potential. *Biophysical Journal*, **7**, pp. 121–35.

Schulz, G. E. and Schirmer, R. H. (1978). *Principles of protein structure.* Springer-Verlag.

Seshu, R., Rendell, L., and Tcheng, D. (1988). Managing constructive induction using subcomponent assessment and multiple-objective optimization. *Proc. International Workshop in Change of Representation and Inductive Bias–I.* pp. 293–305.

Slagle, J. and Gini, M. (1987). Pattern Matching. In *Encyclopaedia of artificial intelligence* (eds S. C. Shapiro and D. Eckroth) pp. 716–20. Wiley Interscience.
Smith, L. M. (1987). Automated DNA sequence analysis: guide to biotechnology products and instruments science. *Nature*, **235**, No. 11, G89.
Sternberg, M. S. E. (1983). The analysis and prediction of protein structure. In *Computing in biological science* (eds Geisow and Barrett), Elsevier Biomedical Press.
Taylor, W. R. (1986). The classification of amino acid conservation. *J. Theor. Biol.*, **119**, pp. 205–21.
Thornton, J. (1986). Loop regions in proteins: their structure, prediction, and antigenicity. *Proc. SERC Collaborative Computational Project in Protein Crystallography (CCP4)*, Daresbury.
von Heijne, G. (1988). Getting sense out of sequence data. *Nature* **333**, pp. 605–7.

20

Design of Knowledge Processing Systems—Principles and Practice

S. Ohsuga

Research Center for Advanced Science and Technology,
University of Tokyo, Japan

Abstract

In this paper we discuss some ideas for making intelligent computer systems really useful and a system developed based on the idea. Key issues in designing such a system are the knowledge representation language and system management. The issues relate closely to the manner of solving problems. The analysis of problem-solving processes by human beings suggests to us that a model-based method must be adopted as a new generic method of problem-solving by computers. The method involves operations to build and modify the model as well as capabilities for model analysis and evaluations. The knowledge representation language must be designed so as to be suitable for representing both the model itself and the model building-analysis-manipulation process. The system must also have the system management capability to define the best scope of knowledge relevant to solve each domain-specific problem and to maintain the system consistency. In this paper we discuss a general view for the design philosophy of such an intelligent system and an implementation.

1. INTRODUCTION

Expert systems have been developed as an embodiment of artificial intelligence (AI) (Hayes-Roth 1983) and it expected that they will enable us, by means of inference mechanisms, knowledge bases and search methods, to solve problems in a most different way from conventional information processing. In reality, expert systems currently in use are applicable only to a limited class of rather small-scale problems. Moreover, they are not always powerful enough to cope with some type of applications. For example, engineering designers hoped that AI technologies would have substantially advanced CAD (Ohsuga 1985b) but current expert systems are rarely powerful enough to meet their needs. In this paper we investigate the reason.

A number of conditions must be met to make knowledge processing

systems useful. We have analysed problem-solving processes by human beings in many problem domains. In this paper we discuss ideas for making intelligent computer systems really useful irrespective of the types of application. Then we present a system developed based on these ideas. Hereafter in this paper, we use the term 'knowledge processing system' rather than 'expert system' because we wish to consider new information processing systems in general.

One of the key issues in designing general-purpose, intelligent systems is knowledge representation. It relates closely to the manner or the style of solving given problems. Analysis of problem-solving processes by human beings suggests to us that model-based methods must be adopted for complex problem solving (Brodie *et al.* 1984). Generally speaking, a model is composed of stuctural information together with a set of descriptions of the attributes, properties, functions, and so on, of the model or its components, called 'functionalities' in this paper. Model-based methods of problem-solving involve operations to build and to modify the model as well as capabilities for model analysis and evaluation. Thus the knowledge representation must be suitable for this.

Model-based methods also require a powerful system management capability. Search for relevant knowledge for given problems occupies a large part of the processing time. Restricting the scope of the search effectively is the role of the management system. It applies meta knowledge to problem-solving at the object level (Davis 1980).

We have designed a system named KAUS (Knowledge Acquisition and Utilization System) embodying these ideas (Yamauchi and Ohsuga 1985). We have analysed the requirements for a knowledge representation language and developed a language to meet them (Ohsuga 1983 and 1985a). This is characterized by 'predicate logic + data structure'. An inference algorithm has been defined in this language. The overall management system is not yet implemented but is under way. We show in this paper that this system is suited for solving a wide class of problems including those requiring trial-and-error processes for goal-seeking.

This paper is composed of two parts. Section 2 discusses a way to make intelligent systems really useful. In Section 3 KAUS is presented as an embodiment of those ideas.

2. KNOWLEDGE PROCESSING SYSTEMS

In this section we consider the concept and some important aspects of knowledge processing systems.

2.1. Formalizing the problem-solving process

As the first step we investigated the problem-solving processes of human beings in various domains and defined them as composed of the model-

building sub-process and the model evaluation sub-process as shown in Figure 1. This figure shows a general framework: specific processes may differ from one another. For example, for the problem to which a model-building method has already been applied, the model-building sub-process may degenerate.

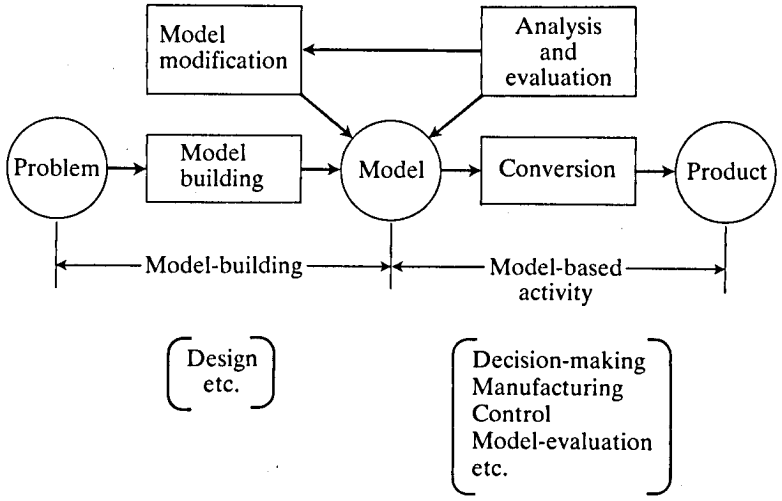


Figure 1. Formulation of the problem-solving process.

The process begins with the problem, see Figure 1, left. The problem solver makes an initial model and analyses it to obtain its functionalities. If the results show that the model is not a faithful representation of the problem, then the problem solver modifies the model and repeats the process. When all conditions are met, the model-building process ends and the problem solver moves to the model evaluation process. During this process information may be added to refine the model as shown in Figure 2.

The model-building sub-process is defined more clearly for synthetic problems, see Kalay *et al.* (1987). Here, the model is a chunk of information expressing various aspects of objects in terms of which the problem is defined. They are classified into two classes by the form of information: one representing object structure and another representing functionalities. Model structure is represented by a data structure while functionalities need to be described in the form of predicates. These two must be connected tightly in the model representation. Then the problem type is classified by the types of operations to the model: the analytic problem is to derive the requested functionalities for the given model structure while the synthetic problem is to obtain the model

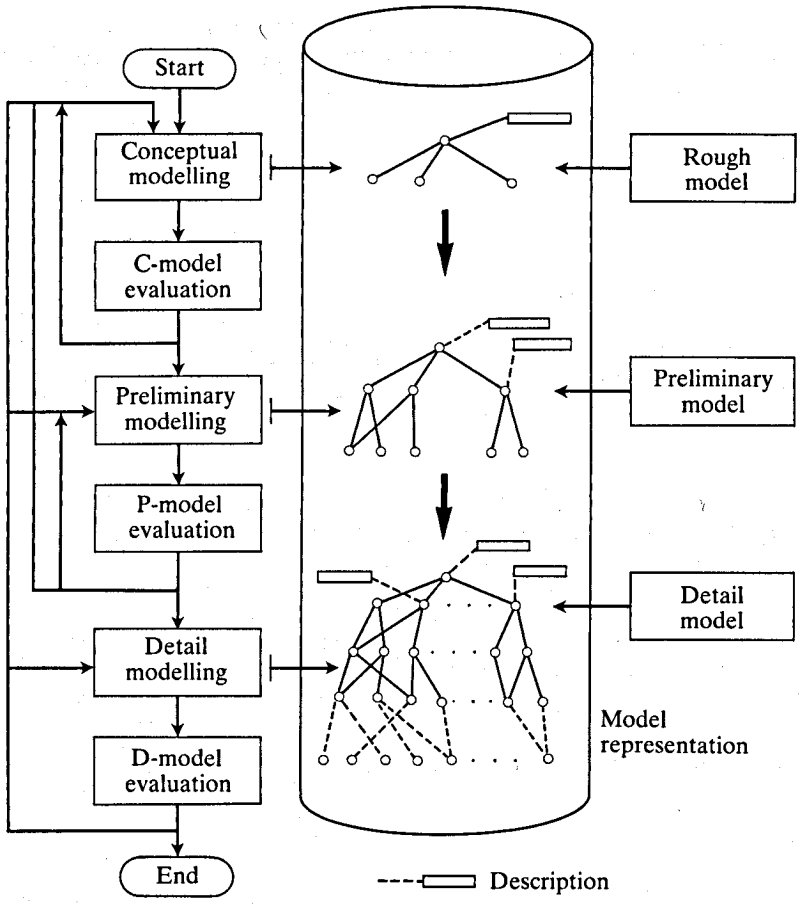


Figure 2. Incremental process of model refinement.

structure that meets the given requirements represented in the form of functionalities.

2.2. Enhancing problem-solving capability

Our objective is to enhance the computer's processing power to aid this problem-solving process. There are a number of possibilities. Among them the following two are expected to be most effective;

(1) to automate as many individual operations (shown by the block in Figure 1) as possible

(2) to integrate these different operations so that any problem-solving process proceeds mostly autonomously in transferring the job from one operation to the next (Ohsuga 1985b).

To fulfil the first objective, we need to give computers the capability to use different methods to achieve the goal and to deal with different forms of information for the purpose. For example, the object model must be analysed and evaluated in many aspects depending on the requirements involved in the problem. For some aspects, analytical methods are established and specific programs, such as structural analysis programs, are available.

For other aspects we need to use data bases directly for model evaluation. In other cases there is no analysis-and-evaluation method other than the engineer's experience. Thus the available information is different depending on the aspect of the model. Knowledge processing technology has to adapt to this situation.

In order to achieve the second objective, various capabilities are required. First of all, the model must be represented in the system in such a way that various methods as above are applicable directly to the model. Very often different information and different methods must be used together to achieve model analysis, evaluation, and modification in the same system: i.e. different operations must be integrated for the same purpose. For example, man's experience must be applied to the model in combination with the data in data bases.

Though the real problem-solving process is different for each specific problem in the general framework of Figure 1, the model analysis subprocess is involved in every case. The use of conventional computers is necessary in analysing the model, and some capability for automatic programming is required. Currently program synthesis is exclusively a manual task. The main difficulty involved in program generation is in most cases that of problem-solving.

Figure 3 shows a simple example. Let us assume that we are going to design a control system. A tentative model is made (Figure 3a) and we wish to know its characteristics. We need the computer to generate what we want as a transformation of the model information. In this example, the object model is first transformed to a signal flow graph (Figure 3b). Then the sets of all local loops and open paths are derived (Figure 3c) with their respective local transfer functions. The global transfer function of the control system can be obtained by applying a mathematical operation to the sets (Figure 3d). This method is Mason's Gain Rule (Gene 1982). Then the response function can be obtained, given an input function, by means of a mathematical formula processing system such as the inverse transformation. Different outputs such as root locus can be obtained too without difficulty after the global transfer function is obtained. This is an example that has been successfully executed by KAUS using different knowledge sources and a mathematical formula manipulation system.

Various analysis methods for obtaining functionalities of the given

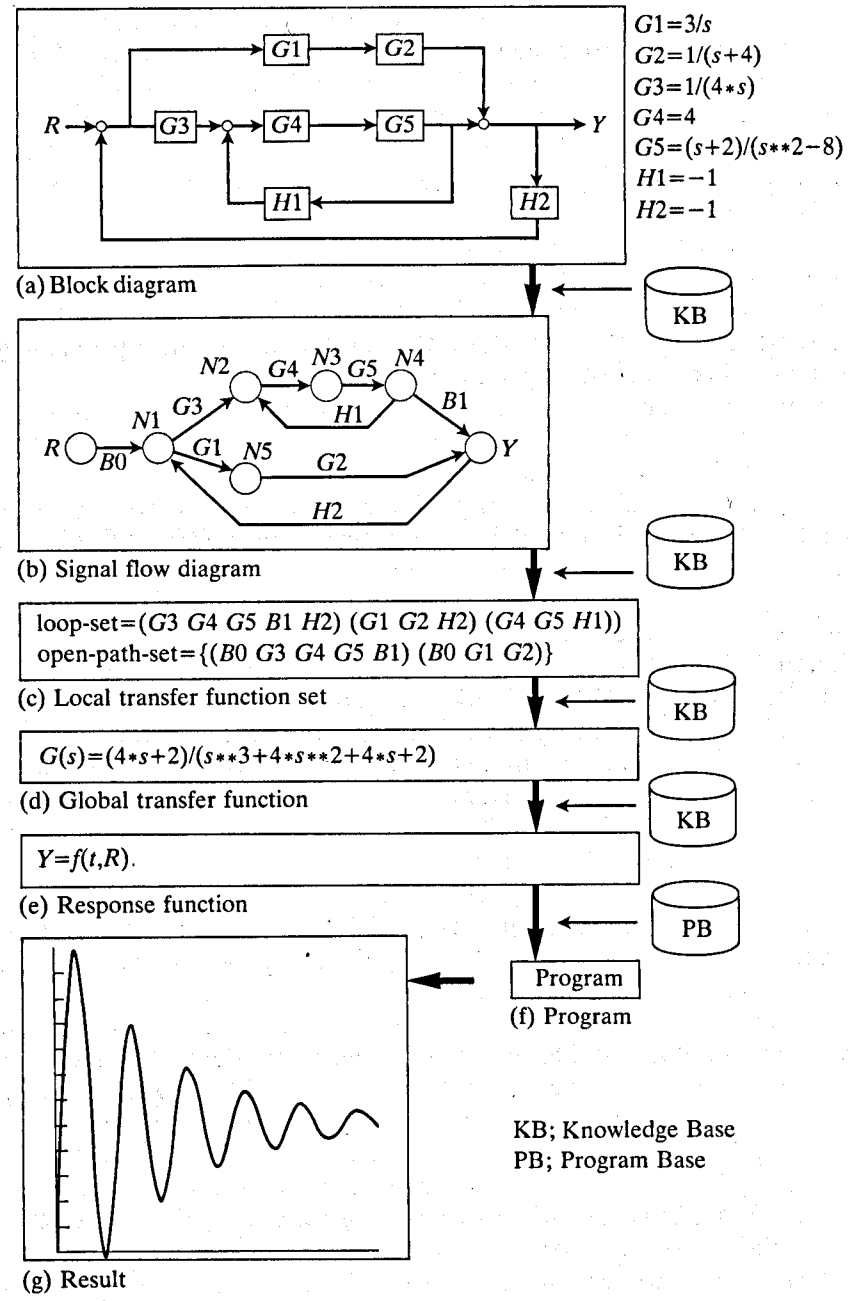


Figure 3. An example of problem-solving.

model structure such as Mason's method in this example have been studied and accumulated in each problem domain. These methods are used effectively for problem-solving by a human problem-solver. This example shows that knowledge processing systems must be able to represent and use these methods. To use them together with human experience, intuitions and so on makes for effective problem-solving.

From these discussions we conclude that knowledge processing systems must be provided with a language that is at a higher level than that of an ordinary programming language. It must be able to describe any kind of model and any type of method for analysing, evaluating, and modifying models in different forms such as procedures, data bases, and man's experiences. The system must also be provided with mechanisms for processing any expression in this language. This language is the knowledge representation language and its processor is an inference mechanism.

2.3. Knowledge representation

One of the key issues in model-based problem-solving is knowledge representation.

First of all it must be suitable for building and manipulating a model to look for the solution of the problem. At the same time, there must be very smooth man–machine communication. The requirements are listed in Table 1 (Ohsuga 1985a).

Table 1. Requirements for knowledge representation.

(a)	Requirements for modelling capability
(1)	Automated consistency checks
(2)	Declarative forms
(3)	Complete deductive inference
(4)	Complex model representation
(5)	Dynamic model building/manipulation
(6)	Ability to handle a large amount of data
(b)	Requirements for smooth man–machine communication
(7)	Declarative forms
(8)	Complete deductive inference
(9)	Conversion to internal language
(10)	Acceptance of external languages

2.4. Management of knowledge processing

A good knowledge representation language is essential in ensuring that the system achieves these goals. The model-based method of problem-solving includes operations to modify the model. Model analysis and

evaluation are the operations necessary to derive a requested functionality of the given model. The functionality is consistent with the model structure in the sense that the structure has the functionality. On the other hand, the model modification is the operation required to modify either the structure or some functionalities of a consistent model. It therefore breaks the consistency of the model and asks the operator or the system to recover this consistency. A single modification operation may induce a sequence of analyses and evaluations to recover consistency for the modified model.

Let us assume that the knowledge processing system is provided with a management system that always keeps the model consistent. Then the analysis-evaluation operations are under the control of this system. On the other hand, the modification operation needs to activate the management system to recover the consistency of the model. Thus modification is an operation at a higher level than that of analysis-evaluation. This means that intelligent systems operate at different levels.

Various models are generated during the model-building process. Each model must be managed separately by the management system. Generally speaking the task is to define and manage the different chunks of information in a knowledge base. It is desirable to give descriptions of each chunk. This is meta-level knowledge. Similarly it is possible to define meta-meta-level and so on. Managing these levels is an important role of the management system.

3. KAUS—A KNOWLEDGE PROCESSING SYSTEM

We have designed a knowledge processing system named KAUS (Knowledge Acquisition and Utilization System) along these lines.

3.1. 'Predicate logic + data structure' as the knowledge representation language

A knowledge representation language is designed to satisfy the requirements listed in Table 1. To satisfy all requirements for man–machine communication ((7), (8), (9), (10)) as well as some for the modelling ((1), (2), (3), (6)), we decided that the language must be based on predicate logic (Chang 1972). In order to satisfy the remaining requirements ((4), (5)) for modelling, it must be expanded to involve various data structures as arguments of predicates. This has been achieved in two steps:

(1) defining data structure (type)

(2) introducing it into predicate logic.

These issues are discussed in the following two sections.

This language can express knowledge in the object-oriented way, that

is, the object structure is defined and descriptions are given to this object.

3.2. Data structure as a generalized set

In order to define such language it is necessary to make clear the concept of data structure in a declarative language. Conceptually it must be very different from that of an ordinary (procedural) programming language because of the difference in the semantics between these languages. Any data structure in a declarative language corresponds to a structure of a real entity in the world being described. In other words, a data structure defined in a declarative language must be such that any real structure corresponds to one of the data structures defined in the language. What is such a set of data structures? We introduce the idea of axiomatic (Zermelo–Fraenkel) set theory (Krivine 1971) in KAUS in defining such a set. The theory says that a largest set of entities free from any contradiction must be defined in a constructive way by using a finite set of primitive constructors (to define a set otherwise can cause contradictions such as that known as Russell's paradox). The term 'set' is almost equivalent to 'data structure' from the point of view of information processing.

A set of the primitive structural relations including 'element-of ($\in$)', 'component-of ($\lhd$)', 'power-set-of (*)', 'product-set-of ($\times$)', 'union-of ($\cup$)', 'intersection of ($\cap$)', and 'pair-of ($\langle\ \rangle$)' are provided with KAUS. These are the structural relations and at the same time work as the structure constructors if one of the entities that should be in the given relation is not yet defined. The set of constructors has been defined with reference to the theory but with slight modifications. For example, 'component-of' is added to represent a hierarchical structure existing in the real world that has no property inheritance.

Any structure constructed as an arbitrary compound of the above is allowed in KAUS. The set of all entities including the given primitives and those that can be generated from them forms the universe. Hence the universe of KAUS involves every structure that can appear in mathematics as well as the hierarchical structure in the real world. The definition of these set-constructors is presented in Table 2.

Note that any element of a set can also be a set. Thus the set is itself a structure. For example, a set $*n\,X$ is obtained by applying the power-set constructor repeatedly n times to the given set X, that is, $*n\,X = *(*n-1\ X)$. This represents a set of all n-level hierarchical structures that can be constructed from the set X. Any specific hierarchical structure constructed from X is an element of the set.

Various useful data structures or relations are defined as the compounds of these primitive constructors. Let © denote a compound operator. Some examples are shown in Table 3. In these structures, hierarchical structure and graphs are very frequently used.

Table 2. A set of primitive constructors.

Constructor	Symbol	Definition
(1) element-of	$\in$	$x \in X$ means that an entity x is an element of a set X.
(2) component-of	$\lhd$	Let X be assembled of the parts $y1, y2, \ldots, yn$. Let Y be the set of the parts, that is, $Y=\{y1, y2, \ldots, yn\}$. Then, $X \rhd Y$.
(3) power-set-of	*	$Y=*X$ denotes that Y is the set of all subsets of a set X.
(4) product-set-of	$\times$	$Y=X1 \times X2 \times \ldots \times Xn = \times Xi$ denotes a Cartesian product of Xi, $i=1, 2, \ldots, n$.
(5) union-of	$\cup$	$Y=X1 \cup X2 \cup \ldots \cup Xn = \cup Xi$ denotes a union set of Xi, $i=1, 2, \ldots, n$.
(6) intersection-of	$\cap$	$Y=X1 \cap X2 \cap \ldots \cap Xn = \cap Xi$ denotes an intersection of Xi, $i=1, 2, \ldots, n$.
(7) pair-of	$\langle\,\rangle$	When $X1$ and $X2$ are the sets, then $y=\langle X1, X2\rangle$ is an allowable entity.

Table 3. Compound structures

Structure	Symbol	Definition
(1) subset-of	$\subset$	$X \subset Y = X \in (*Y)$ or $\subset = \in \copyright *$
(2) part-of	$\lhd$	$yi \lhd X$ denotes yi is a part of X. Then $yi \lhd X = yi \in Y$ *and* $Y \lhd X$, or $\lhd = \in \copyright \lhd$.
(3) table-of	T	A table T is defined as a set of n-tuples. Then Table $(Xi, X2, \ldots, Xn) =$ $\subset \copyright \times Xi = \in \copyright * \copyright \times Xi$.
(4) graph-of	G	A graph is defined as the compound of a pair of tables; a table of nodes and a table of edges. Let N and E be the tables of nodes and edges respectively. Leg G be a graph. Then $G \rhd N$ and $G \rhd E$ where $E = \in *(N \times N)$

3.3. Predicate logic based on the set concept

Data structure is involved in predicate logic by expanding the syntax of many-sorted logic, a branch of ordinary first-order predicate logic, (Enderton 1972) and by including data structure in the definition of the term. It means that a functionality is decribed by the predicates on an object that is represented by the data structure (Ohsuga 1983 and 1985a). With this logic a predicate is represented in the form of $(\forall x/X)F((x)$ or $(\exists x/X)F(x)$ where X is a set in the universe and x/X denotes $x \in X$. $(\forall x/X)F(x)$ and $(\exists x/X)F(x)$ are read 'for all x in X, $F(x)$' and 'for

some x in X, $F(x)$' respectively. These expressions are the equivalent of expressing $(\forall x)\ [x \in X \rightarrow F(x)]$ and $(\exists x)[x \in X \cap F(x)]$ respectively in ordinary predicate logic. X can be any set and therefore can be the generalized set defined above. Thus x can be an individual structure and this predicate gives an expression on the structured object. Or, in other words, this predicate logic involves an object-oriented data type. This is especially important from the practical point of view because it can represent functionalities on any hierarchical object. As its natural extension, it is possible to represent functionalities on some component of given hierarchical objects. For example, an expression such as $(\exists x/X)(\forall y/x)F(y)$ is possible, meaning 'for all components y of some component x of X, $F(y)$'. Or, let X be a power set of the given set W. Then $(\exists X/{*}W)\ (\forall x/X)F(x)$ requires the system to obtain a subset X of W such that every element x of X satisfies $F(x)$. Here $*W$ denotes a power set of W. This is the basic form of giving a constraint to the candidate set W to filter out the set X of such elements x that meet the given condition. We call the system Multi-Layer logic or MLL for short. We show in the following a few examples of MLL expressions:

[A] *Representation of basic mathematical concepts;*

1. The maximum number in the given set;

   $[\forall X/{*}\text{int}]\ [\forall Y/X]\ ((\max X\,Y) \leftarrow ((\forall N/X)\ (Y \geqslant N))$
   where *int means the power set of integer and, therefore, $X/{*}$int means $X \subset$ integer. Note that a variable (X in this example) can be the domain of the other variable (Y).

2. The subset of the other set;

   $[\forall U/{*}d]\ [\forall V/{*}d]\ ((\text{subset}\ U\,V) \leftarrow ([\forall X/V]\ [\exists Y/U]\ (X = Y))$
   is used to examine whether a set (V) is a subset of the other set (U).

3. The intersection of the given sets;

   $[\forall X/{*}d]\ [\forall Y/{*}d]\ [\forall Z/{*}d]$
   $((\text{intersection}\ X\,Y\,Z) \leftarrow (([MU/Z)\ ([\exists S/X]\ (U = S), [\exists T/Y]\ (U = T)))$
   where the symbol M is used to represent a special meaning. It is used in the form of $[MS/X]\ (p\ S)$ to mean that X is the set of all S's that meets the condition $(p\ S)$.

These are special examples to show the expressive power of KAUS. It is to be noted that the inference mechanism of the KAUS system described below can cope with these expressions directly. It is also to be noted that no recursive form is used in expressions to represent such mathematical concepts as those above which usually need recursive form. This is an important condition for a language from the user-interface point of view. For we do not usually use recursive expression in the real word. It is the

role of computer systems to transform these expressions into recursive form for the purpose of processing by machine, if necessary.

[B] *Representation of an analysis method—an example of path-finding for problems represented in graph form*

In order to aid problem-solving, systems must be able to represent many established analysis methods developed in each problem field. The following is an example used to obtain the path for problems represented in graph form.

> [$\forall G$/graph] [$\forall Iv$, $\forall Fv$/vertex] [$\forall$Path/*vertex] [$\forall D$/*int] [$\forall N$/int] ((path G Iv Fv Path) ← (number Path N D), (element-of Path 1 Iv), (element-of Path N D), [$\forall N2/D$] [$\exists$Arc/G*$g-a$] [$\exists N3$/int] ((is $N3$ $N2+1$), (element-of Path $N2$ Arc:ip), (element-of Path $N3$ Arc:tp)))

where (path G Iv Fv Path) denotes that (an ordered set) Path is the path starting from the starting point Iv to the goal point Fv in the graph G; (number Path N D) denotes that the Path corresponds to the set D of N elements; (element-of Path $\dot{X}$ Y), denotes that the X-th element in the ordered set Path is Y respectively. Arc:z is a special syntax to refer to z components in the hierarchical structure Arc. The body of this formula says that the path is the ordered set of nodes in which adjacent elements are the starting point and the end point of an arc. To represent concepts defined in graph theory is quite important because many problems are solved after being translated into a graphical form.

Thus the KAUS knowledge representation language can correspond directly to various concepts included in the expression in the external language such as natural language and picture drawing and it is also well suited for representing various kinds of models and their manipulations.

3.4. Inference of MLL

There is the following equivalence rule;

$$X \supset Y \quad \leftrightarrow [(\forall x/X)F(x) \rightarrow (\forall x/Y)F(x)]$$
$$X \subset Y \quad \leftrightarrow [(\exists x/X)F(x) \rightarrow (\exists x/Y)F(x)]$$
$$X \& Y \neq \emptyset \leftrightarrow [(\forall x/X)F(x) \rightarrow (\exists x/Y)F(x)]$$

These relations are used for inference. Unification by symbol manipulation can be replaced by the test for set-theoretical relations between the domain sets. As any compound structure can be decomposed to elementary sets, the set-theoretical relation as above can be decomposed to a set of set-theoretical relations between elementary sets. A few examples of decomposition theorems are given below:

(1) $X \subset Y \leftrightarrow {}^*X \subset {}^*Y$
(2) ${}^*(X \cap Y) \leftrightarrow {}^*X \cap {}^*Y$

(3) if $\cup X \neq X$, then $X \subset^* (\cup X)$
(4) if $\cup X \neq X$, then $\cup X \subset Y \leftrightarrow X \subset^* Y$
(5) if $\cup X \neq X$, then $X \cap^* Y \neq \emptyset \leftrightarrow \cup X \cap Y \neq \emptyset$

The other part of the inference method is the same as standard resolution.

3.5. Knowledge structure composed of the relations between entities

To aid the test for the set-theoretical relation between elementary sets, a conceptual entity network is formed. Every entity defined to the system is linked to every other by primitive structural relations to construct a conceptual entity network. Then the test is performed by traversing the network in a specific way.

Every logical predicate is embedded in the network in such a way as to link predicate entities to network entities. This forms a knowledge structure.

This network is also effective defining an arbitrary local world. A local world is defined as being composed of a subset of entities in the universe and a set of logical formulas that includes the entities in this subset or in a region specifically related with it as the arguments. The mechanism for defining the local world is itself a data structure but is separated from the object level structures. This mechanism is managed by the management system and forms the multi-level organization.

3.6. Object model

The object structure and its descriptions are amalgamated in order to represent a model in KAUS. A model forms a local world in the knowledge structure. An example of a model representation is shown in Figure 4.

Any predicate can include structured objects as the arguments. The predicate is linked to the root node representing the object structure. At the same time, the structure can include the other predicates linked to the lower-level nodes as the structural constraints or describe the functionalities of the sub-structures represented by these nodes. Thus the predicates involved in this structure are related to each other indirectly through the structure. This facilitates the handling of constraint propagation. The fact that a model is a part of the knowledge structure means that the amalgamation of object structure and its descriptions is a particular feature of the KAUS knowledge representation (see Figure 4).

3.7. Integration of resources in KAUS

The knowledge representation language of KAUS also plays an important role in integrating the various resources in the system. As has been

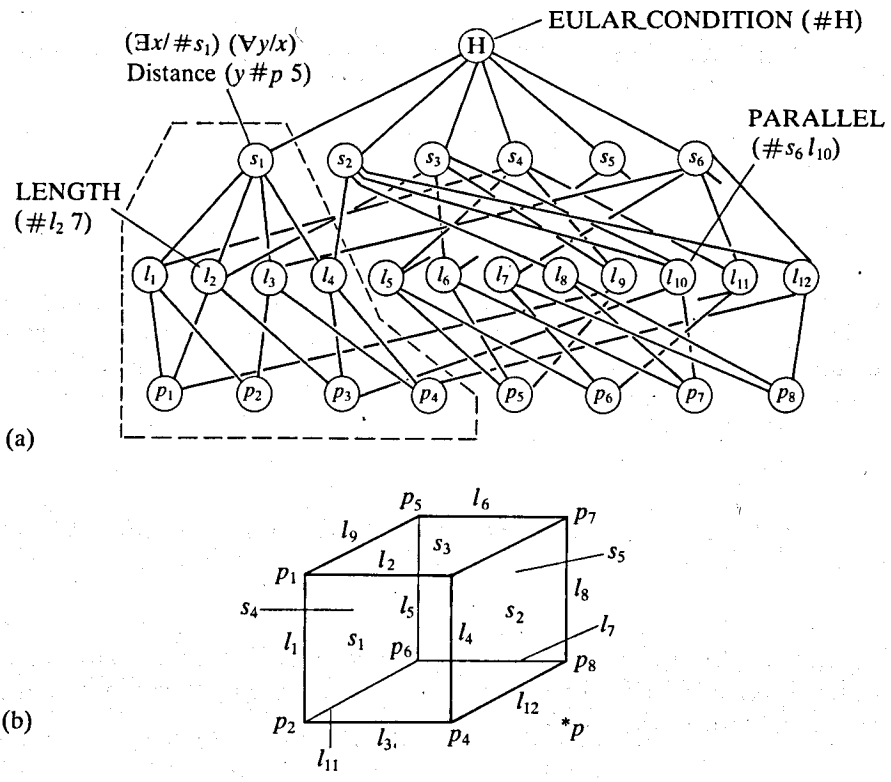


Figure 4. Model representation.

mentioned in the requirements for the knowledge representation, knowledge processing must be used together with conventional information processing. In fact, most of the real operations on the model are executed by conventional programs.

3.7.1. *Use of existing programs*

KAUS has a set of built-in predicates called procedural type atoms (PTA). Each PTA has a corresponding procedure in the procedure base which is evoked by the inference mechanism when a certain condition is met. Then it is executed to return the value(s) of some variable(s) as the result. Each PTA is represented by the procedure name followed by the input and output variables included in the program. KAUS is ready to accept any kind of procedures defined by the user with its PTA.

Only specifically written programs can be used with this method. For transferring control between the knowledge level operation and the programs developed independently with KAUS, another way of linking is provided. For example, in a version of KAUS running under Unix a special

PTA named 'exec' is used. The exec-PTA is a second-order predicate and accepts a program name as an argument. It interprets the first argument as the program name and the remainder as the arguments of this program. It uses the fork-and-execute mechanism of Unix. It evokes a specified program and receives the result. Then it returns to the inference mechanism either 'yes' or 'no' according to the end code of the program. Thus it is possible to use every program working under the Unix system. A set of these programs forms the program base.

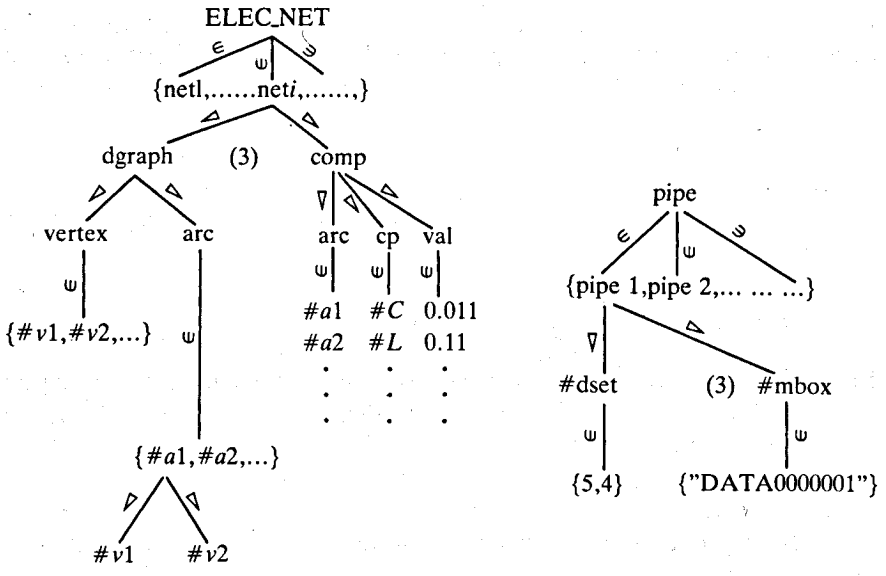
3.7.2. *Integrating data bases with knowledge bases*

Integration of knowledge bases and data bases is a particularly important issue (Gallaire *et al.* 1984). In order to deal with a large set of data to represent a model structure, the use of data bases is mandatory. Data bases used for this purpose must be able to store the structured object. We need a data base with a specifically designed data model. Moreover, knowledge processing systems must be able to communicate with the existing DBMS in order to use the data in the existing data bases. Thus KAUS is provided with two types of data bases with different couplings: tight coupling and loose coupling. For the former, any access method can be defined at the knowledge level and executed by the inference mechanism incorporated with a PTA mechanism. The data model is expanded so as to represent complex data objects.

An example is shown in Figure 5. Figure 5a shows a knowledge level data structure representing a set of graph structures, each of which is composed of two table-type relations in the data base to represent a set of nodes and a set of arcs respectively. In this figure, the root node represents the set of graph structures, net*i*, defined in the data base. The graph also has some attributes attached to each node or each arc which are shown stored in the 'comp' relations in Figure 5a. Object model structures in many KAUS applications are, in reality, represented in the data base.

For the purpose of communicating with the conventional processing level, the data structure of Figure 5b is used in the KAUS knowledge base. It is composed of the pipe descriptor for communication and the name of the file to be used for data exchange with the data base. We call this data structure the 'port'. There is a set of ports in the knowledge base to communicate with different data bases.

As the existing DBMS cannot be modified for the purpose of integration, the loosely-coupled method becomes necessary. KAUS generates the access commands to these data bases. The same data structure as in Figure 5 is used. KAUS knows, referring to the data structure, the relation schema and data base schema of the data base and generates an intermediate code using this information. Then the code is translated to the



(a) Representation of complex object model

(b) Representation of communication ports

Figure 5. Media for knowledge-base and database communication.

commands each DBMS requires. Now KAUS is able to access two different DBMSs as well as generating SQL commands.

In either communication, KAUS is the master and the DBMS is the slave. What we wish to stress here is the fact that this type of integration became possible because the knowledge representation can deal with the data structure. This is quite important because, otherwise, the communication method cannot be represented in the knowledge base and, as a consequence, the knowledge processing part must be the slave in the system.

3.8. User interfaces

A good user interface is always important (Fitter 1979). However, it is particularly important for knowledge processing systems. A user interface has been implemented in KAUS to allow a mixture of various media such as a subset of natural language, mathematical formulas, and picture drawing on the multi-window system. Figure 6 is the interface used in the example shown earlier in Figure 3.

3.9. Examples of KAUS applications

KAUS has been applied to various problems in order to test its applicability. Figure 3 (above) shows one of these applications. It represents a

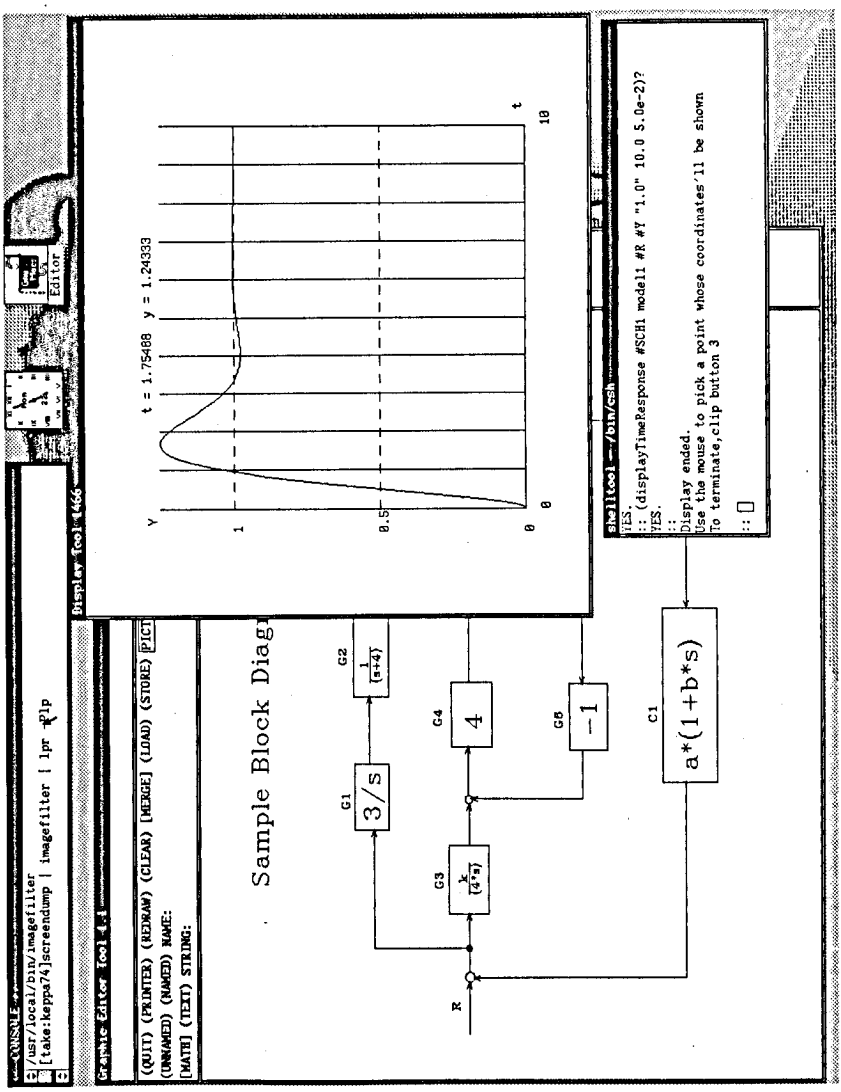


Figure 6. An example of the user interface.

control system design. A designer makes a tentative model on the display using both pictures and mathematical formulas. (We have designed a natural language interface but it is not necessary for this example). The block diagram represents an object model in this applicaiton because it is comprehensible to man and involves all information necessary to achieve the goal. The object model is transformed to the signal flow graph. The latter is a model representation transformed from the original model for the purpose of analysis, to which the existing method for analysing this diagram (e.g. as Mason's method) can be applied directly. Then, as discussed before, the sets of local loops and open paths are derived with their respective local transfer functions. A global transfer function for the control system can be obtained by a specific mathematical operation on the sets. The mathematical formula processing system REDUCE has been used. Then various characteristics of this control system can be obtained. The transformation methods and the analysis methods are represented in the form of knowledge and are used by the inference machine. The analytic results are displayed as shown in Figure 6. This is, however, a very simple example because a completely analytic method is available to obtain the solution. Nevertheless it is very useful even in a synthetic problem, because the designer can evaluate his model rapidly and make the next decision. This accelerates the synthesis process.

In addition to the above example, the system has been applied to more complex problems including the electronic circuit analysis, mechanical design (Han *et al.* 1987), and the structure estimation of chemical compounds. Some of the applications need to combine human experience with established analysis methods. These applications confirmed the utility of the ideas discussed in this paper.

4. CONCLUSION

We have discussed some basic ideas for ensuring that intelligent systems are really useful and then presented an implemented system based on the ideas. This system, named KAUS (knowledge acquisition and utilization system) has shown a capability in processing information that has not been achieved by conventional information processing.

Currently only the object level subsystem of KAUS has been developed and is running. The management system, including multi-level control and many-world control, is being implemented.

Acknowledgements

The author would like to acknowledge the contribution of Mr H. Yamauchi, Mr. T. Akutsu, Mr A. Takasu, and Mr J. B. Guan in implementing the KAUS system.

REFERENCES

Brodie, M. L., Mylopoulos, J., and Schmidt, J. W. (eds) (1984). *On conceptual modelling—perspectives from artificial intelligence, databases and programming languages,* Springer-Verlag.

Chang, C. L. and Lee, R. C. T. (1972). *Symbolic logic and mechanical theorem proving,* Academic Press.

Davis, R. (1980). Meta-rules: reasoning about control. *Artificial Intelligence* **15** No. 3, pp. 179–222.

Enderton, H. B. (1972). *Mathematical introduction to logic,* Academic Press.

Fitter, M. (1979). Towards more natural interactive systems, *Internat. J. Man-Machine Studies* **11**, pp. 339–50.

Gallaire, H., Minker, J., and Nicolas, J. M. (1984). Logic and databases; a deductive approach, *Computing Surveys* **16**, No. 2.

Gene, H. H. *et al.* (1982). Design of feedback control systems, Holter-Saunders.

Han, G., Ohsuga, S., and Yamauchi, H. (1987). The application of knowledge base technology to CAD. In *Expert Systems in Computer Aided Design* (ed. J. Gero), North-Holland, pp. 25–55.

Hayes-Roth, F. *et al.* (1983). *Building Expert Systems,* Addison-Wesley.

Kalay, Y. E., Swerdloff, L. M., and Harfmann, A. C. (1987). A knowledge-based computable model of design, Preprints Working Conference W.G.5.2 on *Expert Systems in Computer-Aided Design.*

Krivine, J. L. (1971). *Introduction to axiomatic set theory,* D. Reidel Pub. Co.

Ohsuga, S. (1983). A new method of model description—use of knowledge base and inference, in *CAD System Framework* (eds K. Bo and F. M. Lillehagen), North-Holland, pp. 285–312.

Ohsuga, S. (1985). Multi-layer logic—a predicate logic including data structure as knowledge representation language, *New Generation Computing,* **4**, (Special issue on knowledge representation), Ohmsa Ltd. and Springer-Verlag.

Ohsuga, S. (1985). Introducing knowledge processing to CAD/CAM, *Finite Elements in Analysis and Design,* **1**, pp. 255–69.

Yamauchi, H., and Ohsuga, S. (1985). KAUS as a tool for model building and evaluation. In *Proc. 5th International Workshop on Expert Systems and Their Applications,* Avignon, France, May 13–15.

INDEX

Note: illustrations and captions are indicated by *italic page numbers*, footnotes by suffix 'n'.

absorption operator, DUCE's use of 94, 97, 176
abstraction
  qualitative modelling at various levels of 259–79
  with respect to a constant 85
ACLS algorithm 197, 206
active learning, data acquired by 170
actor-oriented coordinate frame
  advantages of 237
  decision tree for 232, *233*
  object-pushing experiment 232
Ajdukiewicz's solution to name relation paradox 85–7, 88
algebraic semantics 49
  definitions for 50–2
  notation for 50–2
  results using 52–3
algorithms, structure-moving task 209–17
ALG(SIG, *E*) 50
AL/X expert system shell 188
antinomy, meaning of term, 80n
anti-unification 97
AQ families of inductive algorithms 93, 106, 169, 298
archaeological objects, classification of 165
argument-based model (for plausible reasoning) 60–3
  advantages of 75
  compared with probabilistic model 61
  implementation examples 68–74
arguments
  coherence of 62–3
  handling of 61–2
  meaning of term 61
Assistant 86 learning system 221, 230
  knowledge extracted from data by 232, 234
assumption-based truth maintenance system 178
  use in incremental learning systems 178–9
attributes
  effect of typing on 164
  types used in CHARADE 162
augmented truth maintenance systems 14
Autolander problem, C4 compared with Evidencer algorithms 197, 198
axiomatic semantics of specifications 21–3
axiomatic set theory 319
backtracking 173
  diagnosis using 272
  YAPES's use of 75
backward diagnosis 260, 268, 271–2
  cardiac diagnosis using 271–2
  compared with,
    hierarchical diagnosis 275
    naive strategy 275
  diagnostic efficiency of 275
  right-to-left goal selection strategy used 268, 271
backward pruning of plausibility trees *194*, 200
bang–bang control 242
Bayes
  naive 189
  sequential 187–200
Bayesian inference nets 190
  Kononenko's treatment 201–2
Bayesian theory, probabilities defined in 187
beam search, use of 298–9
belief states, compatibility with truth values at run-time 195
belief updating, mechanism in probabilistic calculus 56, 58, 60
benchmark data-sets, induction of trees from 196–7
binary logical attributes, domains with, trees induced from 198
BINEYE program 206, *207*, 210, 228, 229
bird example
  Closed World Specialization algorithm used 112
  generalization in 105
  most-general-correct-specialization used 111
  over-specialization in 107
bisection method 276
blackboard systems 3–4
  knowledge systems coupled by 7
block diagrams, knowledge processing *316*, 328
Boolean combinations of events, rules used by PROSPECTOR for 70
boredom, robot's response to 238
break-points, decision tree 128
broadcasting, knowledge systems coupled by 7
Brookhaven (protein) data base 299, 307

bubble-sort algorithm
generalization of 142
sample computation using 140

C4 rule-induction algorithm 128, 187
compared with Evidencer algorithm 197
example of output 129
software facilities available 135
calculi
certainty 63–74
coupling of 6–7
ExpertPRIZ *15*
knowledge representation by means of 4–6
meaning of term 5
NUT system *13*
plausible reasoning 55
calculus of computable functions 4
CAP (Concept Acquisition Program) 179–83
concept description for 181
concurrent running of learning tasks 183
learning strategy summarized 182
liquid pouring example 183
partial match used 179, 180, 181
'tantrum' by 183
cardiac diagnosis 260–1, 265–76
rules used 251, 267
CART rule-induction algorithm 187, 188, 197
causality, added to pole-balancing problem 246
certainties
calculation of 72–4
comparison of 58–9
non-numerical system of 71–4
PROSPECTOR-like approach to 69–71
certainty calculus
definitions used 63–4
formal apparatus for 63–7
implementation examples 68–74
inference procedure for 65–7
model theoretic semantics for 65
semantics of 64–7
certainty lattice, use in example 71–2
certainty space, definition of 63–4
characteristic sample set
algorithm to generate 100
meaning of term 99
size of 101
CHARADE system
characteristics of 156
description language used 161–5
exploration of the description space by 159–61, 165–6
functionalities of 156
learning bias in 161–6
new descriptors added 165
regularities detected by 157–9, 165
representation formalism for 161
rule properties used 159
semantics for generalization heuristics 160
chemical compounds
structure estimation of 328
see also proteins
chemical theory methods, protein structure predicted using 294
chess positions
encoding of 284, 286
entropy of space of 285–6
patterns stored in memory in relation to 286–7
guessing of 287–8
experimental results for 288
multi-answer questions used 288
questions allowed 287
information content of 283–8
information required to uniquely determine 288
pattern recognition of 284
psychological assumptions used 283–4
chess problems
CIGOL's performance in 106
DUCE algorithm used 94–5
chess-specific knowledge
amount remembered by chess master 288
meaning of term 283
number of patterns stored in memory as measure of 284–5
patterns used 284
Chinese language, number of ideograms in, 288n
chunks (of information)
meaning of term 284
number in short-term memory 284, 286
CIGOL learning system 106, 113, 176
chess problem tackled by 106
classification rule, induction of 121
class probability trees
meaning of term 188
use of sequential Bayes with 187–200
clause entailment, resolution used for 110, 114
clause removal, specialization by 107
Closed World Assumption inference rule 108
closed world specialization 111–13
Closed World Specialization Algorithm, implementation in PROLOG 112–13
clustering algorithm 234, 235
cluster of objects, description of 182
cn2 program 298

coated-steel products, interactive induction used 123–31
coherence, types of 62–3
commongrasp procedure 216
complexity, reduction for pole-balancing problem 246–8
compound structures, used in KAUS 319, *320*
computational models
  all variables calculated for 42
  applications of 44
  calculating programs called from 40, 42, 43
  computational rules for 40
  example ('square') 39, 40–3
  implementation in PRIZ 39, 44
  implementation in PROLOG 43–4
  meaning of term 39–40
  meaning of term in PRIZ system 8
  particular variables computed for 42
  specifications comprising 22–3
  use of 42–3
  writing of 40–2
concepts
  definition of 170
  formation in learning systems 170
  theory as network of 172, *173*
concepts of concepts, formal language for 81, 85
conceptual entity network 323
conditionalization, beliefs updated by 56, 58, 60
CONFUCIUS system 169
consistency, maintaining in incremental learning systems 170, 177–9
control rule, pole-balancing problem 253–4
coordinate frame
  object-pushing experiment 230–2, 237
  choice of coordinate system 231–2
correct-specialization, definition of 109
credit assessment 128
  belief-states modified by subsequent information 195
credit assessment problems, C4 compared with Evidencer algorithms 197, 199

data acquisition, methods used in learning systems 170
database clauses
  meaning of term 261
  qualitative models as hierarchy of 261
data bases, integration with knowledge bases in KAUS 325–6
data structure
  as generalized set 319
  knowledge representation using 318–19
data validation task, interactive induction 133, 134
decidability problem 52–3
decision trees
  breakpoints for 128
  build-up by Quinlan method 157
  coated-steel products case study 126–8
  object-pushing experiment, effect of coordinate frame 232, *233*
deductive system, meaning of term 5
deep knowledge, use of 259, 261
deep models, compared with surface models 264–5
denotation and sense, formal language based on 81
derivation trees, transformation of refutation trees into 114–15
'deriv' function 108, 114
  definition of 114
  as explanation-based generalization technique 116–18
  PROLOG implementation of 117–18
  use of 114–15
description languages
  CHARADE's use of 161–5
  generality elaborated using 155
description space, exploration by CHARADE 159–61, 165–6
descriptors, CHARADE's use of 163–4
details (of model), refinement of 266–7, 277
'diagnose' algorithm
  cardiac arrhythmias detected by 269–70
  equations solved using 276–9
diagnostic task 260
  backward reasoning used 260, 268, 271–2, 275
  generate-and-test strategy used 260, 275
  hierarchy of models used 260, 265–8, 273–4, 275
directed acyclic graph 296
discernment, frame of 56
DISCIPLE system 123
dot operator 114
dots expressions 140–2
  folding-up rule for 142
  generalization rule for 142
  standardization rule for 142
dots strings, unfoldment of 141
dots terms, unfoldment of 141
DUCE algorithm
  absorption operator for 94, 97, 176
  applications of 94–6
  chess problem solved using 94–5
  compared with other expert systems 95
  completeness of operators for 96, 98–101
  description of 93

DUCE algorithm (*cont.*)
extensions to first-order representation 96, 102
first-order version; *see* CIGOL
identification operator for 94, 97
interactive induction using 131
intra-construction operator for 93, 98, 176
inversion of resolution in 97–8
macro-operators for 101–2
neuro-psychology application 95–6, 131, 134–5
noisy data dealt with by 95, 102, 134–5
operators for 93–4, 175–6
search mechanism for 96, 101–2
supervised compared with unsupervised mode 95
theory behind 96–102
validation in 133
dynamic control, use of PANIC rules for 239

EARL system 131
validation in 133
electrical circuits, computation of parameters for 44–7
electrical transformers, diagnosis of breakdown in 131–2
electrocardiographic (ECG) descriptions
diagnosis based on 260–1, 265–76
P-wave considered 267
electronic circuit analysis, KAUS applied to 328
elicited knowledge, combined with induced knowledge 127
engineering design, expert systems used in 311, 328
engineering fault diagnosis, C4 compared with Evidencer algorithms 197, 198
entropy-minimization principle 197, 199
entropy of space
definition of 285
determination of 286
experimental determination of 288
number of chess positions measured 285–6
envisioning 260
equation solving, 'diagnose' algorithm used 276–9
error recovery strategy 177–8
errors
correcting of 174–6
locating of 172–4
recognizing of existence 172
error tolerant learning systems 169–84
Evidencer algorithm
arithmetic in 190
classification tree extracted 194–5
compared with C4 algorithm 197
evidence tree induced 191–2
learning in 189
tree-structured rules used 187
evidence trees
conversion to/from plausibility trees 192–4
induction from benchmark data-sets 196–7
results 198–9
induction of 190–2
set-up of 192–5
exemplar-based learning algorithms 294
expected-weight-of-evidence criterion 190, 197, 199
expenses claims problem, C4 compared with Evidencer algorithms 197, 199
experimentation
learning by 170
pitfalls of 171
run-out of ingredient 182–3, 184
ExpertPRIZ system 4, 14
calculi of *15*
experts, network of 6
expert systems
supervised compared with unsupervised mode 95
use in engineering 311
see also knowledge processing systems
explanation-based-generalization (EBG) algorithm
'deriv' function as 108, 116–18
improvement on 105, 108

first-order formula, correctness of 108
first-order logic, sentences formalized in 79, 80
Fitch form (for natural deductions) 27
FOR-loops, inductive synthesis 144–5
forward chaining rules, CAP's concept description as set of 181
Freddy 3 robotics project
goal of 205
hardware implementation of 206, *207*, 227, *228*
learning by 230–9
software implementation of 206, *207*, 227–30
structure-moving task for 208–20
full computation traces, inductive synthesis from 139

gain ratio adjustment 197, 199–200
GenBank (protein) data base 307

generalization
  construction of rules by 156–7
  description languages used 155
  explanation-based 105, 116
  formal relations used 152
  formulation of 151–2
  from general to specific 157
  from specific to general 156–7
  least general 97, 98
  meaning of term 105, 106
  operators required 175
  protein structure problem 296–7
  testing of 179–80
generalization tree 296
generalized regular expressions
  examples of 145
  inductive synthesis 145–6
  interactive synthesis algorithm constructed for 145–6
generate-and-test problem-solving strategy 260
  compared with backward diagnosis 275
  compared with hierarchical diagnosis 271, 275
  diagnostic efficiency of 275
Gentzen-type sequent version of S4, rules of 32
global transfer functions 315, *316*, 328
GPS planner 225–6
grammars
  interactive induction 125–6
    refinement of 126
graphical expressions, inductive synthesis 146–7
Graph Traverser program 225–6
grasping positions, examples of *216*
greatest common divisor (GCD) algorithm
  generalization of 142
  sample computation using 140
ground specifications 51

HACKER program 173
heart
  abstract model of *265*
  detailed model of 267–8
  more-refined model of *266*
Hebrand base 64
Herbrand Universe 64
Heyting–Kolmogorov interpretation 23
hierarchical diagnosis 268–71
  compared with,
    backward diagnosis 275
    naive strategy 271, 275
  'diagnose' algorithm used 269–70
  diagnostic efficiency of 275
  example of use 273–4
hierarchical diagnostic algorithms 269–70
  bisection method emulated by 276
  equations solved using 276–9
hierarchical type
  tree representation of *163*
  use by CHARADE 162, 163
hierarchy of models 265–8
Hilbert's tenth problem 53
hill-climbing beam search 298
homology-based methods, protein structure predicted using 294
Horn clause logic
  argument-based model in 61–3
  derivation of new facts in 12
  extended to handle uncertainty 63
  negation in 62
  plausible inference and negation in 55–76
  procedural knowledge united with 7–8
  propositional 17
Horn clause logic programs, debugging of 106
Horn clause programming 17
human memory, capacity of 284

ID3 algorithm 106, 126, 128, 169
  inductive learning by *234*
  programs similar to 221, 230, 294
identification operator, DUCE's use of 94, 97
ID families of inductive algorithms 93, 106, 126, 128, 169
if–then rules, evaluation of 69–70
imitation, learning by 172, 179–80
incremental learning algorithm, action of 105
incremental learning systems 169
  correcting of errors 174–7
  locating errors in theory 172–4
  maintaining of consistency in 170, 177–9
  problems in 171–9
  recognizing existence of errors 172
incremental specialization techniques 106–8
independent subtasks
  rules of structural synthesis with 31
  use in PRIZ 30–5
indirect context, concept first introduced 80
individual concepts and propositions
  first-order theories of 79–88
  McCarthy's theory of 82–5
INDUCE algorithm 106, 156
induced knowledge, combined with elicited knowledge 127
inductive concept learning, shift of bias for 153

inductive formation (of programs and descriptions) 91–149
inductive learning
  logical description extracted from noisy data by 221, *222*
  plausibility measures for 76
  protein structure problem 294, 295
  robot control using 227, 234
inductive program synthesis 49
inductive syntactical synthesis
  dots expressions used 140–2
  FOR-expressions in 144–5
  generalizaed regular expressions in 145–6
  graphical expressions used 146–7
  models of 139–47
  program optimization using 147
  WHILE-expressions in 143–4
inference algorithms, synthesis methodology for design of 51
inference procedure
  certainty calculus 65–7
    implementation examples 68–74
inheritance, calculus of 9–11
interactive induction 121–36
  case studies 123–32
  coated-steel products case study 123–31
    process grammars used 125–6
    processing unit selection in 126–31
  data validation task in 133, 134
  grammars used 125–6
  irrelevant attributes suppressed 128
  knowledge acquisitiion in 122, 124, 130
  knowledge engineering aspects 132–5
  man–machine interaction in 134
  neuro-psychology case study 95–131, 134–5
  problem formulation task in 133, 134
  rule presentation task in 133–4
  software facilities for 135
  transformer diagnosis case study 131–2
  validation in 132–3, 134
interpret procedure 215–16
interrupt-revision 189, 195
interview-driven knowledge acquisition methods 122
  disadvantages of 122, 124
intra-construction operator, DUCE's use of 93, 98, 176
intuitionistic propositional calculus (IPC) 4, 8
  depth reduction in 25–6
  disjunction eliminated in 26
  negation and disjunction eliminated in 26
  procedural knowledge united with 8–9, 17, 21
KARDIO expert system 260–1
  backward diagnosis used 271–2, 273–4, 275
  experiments for ECG diagnosis 273
  hierarchical diagnosis used 268–71, 275
  naive (generate-and-test) strategy used 275
KAUS (Knowledge Acquisition and Utilization System) 312, 318–28
  applications of 326, 328
  data bases integrated with knowledge bases in 325–6
  existing programs used by 324–5
  integration of resources in 323–6
  Multi-Layer Logic used 321–2
  object model used 323, *324*
  set-constructors used 319
  structures allowed in 319
  user interfaces in 326, *327*
Kedar-Cabelli–McCarty explanation-based-generalization algorithm 108
Kleene realizability 23
knowledge
  modularity of 3–15
  qualitative representations of 203–79
  stratification of *14*
knowledge acquisition
  applications and models of 281–328
  factors affecting validation method used 132–3, 134
  inductive learning used 76
  interactive induction system 122, 124, 130
knowledge-base bootstrapping 125, 136
  user interface for 126
knowledge bases, integration with data bases in KAUS 325–6
knowledge processing, mechanics of 1–89
knowledge processing systems
  design of 311–28
  enhancement of problem-solving capability 314–17
  formalization of problem-solving process 312–14
  KAUS system described 318–28
  knowledge representation in 317
  management of 317–18
  see also expert systems
knowledge representation 312, 317
  formal calculi as means of 4–6
  in probabilistic calculus 57–9
  requirements for 317
knowledge representation language, PRIZ input using 9–10
knowledge structure, entity relations comprising 323
Kononenko's Bayesian nets 201–2

Laplace's Law of Succession 192
lattice
  examples of use 70–2
  meaning of term 62
learning bias
  derivation from rule properties 151–66
  meaning of term 153
  modification of 153
  nature of 154–5
  phenomenological approach to 154
  role of 153–4
learning set, regularities looked-for in 157–8
learning systems
  error-tolerant 169–84
  optimality and error in 149–201
  robot control using 226–7
  types of 169
least general generalization, concept of 97
light pen, use in inductive synthesis 147
liquid-pouring example
  CAP used in 179–83
  errors introduced by over-generalization 171–2
local transfer functions 315, *316*
logic, procedural knowledge united with 7–9
logical attributes, domains with, trees induced from 198, 199
logic-based representation
  separation from numerical representation 221
  transformation to numerical representation 221, *222*
logic programming
  meaning of term 17
  propositional logic used 17–36
long-term memory
  capacity of 284
  number of patterns stored in,
    as measure of chess-specific knowledge 284–5
    relation to entropy of space of chess positions 286–7
lookinto operator 209, 212–13

McCarthy's first-order theory of individual concepts 82–5
  concept objects represented in 82
  definitions 82–4
  examples 82–5
  real-world objects represented in 82
macro-operators
  DUCE's use of 101–2
  effect of 102
man–machine interaction, interactive induction 134
Marvin system 169, 176
MARVIN system, validation in 133
Mason's Gain Rule 315
Mathematica system 196
measurement errors, effect in structure-moving tasks 220
memory, human, capacity of 284
meta-interpreter, use in PROLOG 43–4
metalanguage, PRIZ system 9–10
meta-level knowledge 318
metatheories, use of 11–13
minimax problem, PRIZ's solution of 23–5
mirror crafting example, learning in 183, 184
MIS program 173, 174, 175
mistake matching rules, protein structure problem 306
MLL (Multi-Layer Logic) 321
  examples of 321
  inference of 322–3
modal logic S4 17, 30–5
  derivability in 32–4
model-based interpretation 205–23
model-building sub-process 313
MODELER program 107
model generation
  learning procedure used 234
  object-pushing experiment 232–6
model refinement, incremental process of 313, *314*
modularity of knowledge 3–15
most-general-correct-specialization (MGCS)
  definition of 109
  resolution-based method for construction of 110–11
most-general-unifier (MGU) 115
motivation, robot experiment 238
multiple evidence, combined by PROSPECTOR 70
multi-valued logical attributes, domains with, trees induced from 198
MYCIN expert system 55

naive Bayes 189
naive Bayes classifier 201
naive strategy; *see* generate-and-test problem-solving strategy
name relation paradox 80
  Ajdukiewicz's solution to 85–7, 88
  Church's formalization of solution to 85, 86
  Frege's solution 80–1
  McCarthy's solution 82–5
negation, semantics for 62
negation by failure 108, 111
neural net learning 188

neural networks, protein structure predicted using 294
neuro-psychology application, DUCE algorithm used 95–6, 131, 134–5
NEWGEM, rule evaluation function in 298
noise
  errors caused by 177, 178
  multivariate domains with, tree induced from 199
Noiseless Coding Theorem 286
noisy data
  DUCE algorithm adapted to deal with 95, 102
  extraction of logical description from 221, *222*
NONLIN planner 225–6
non-monotonic formalisms 108
  over-specialization avoided by 111–13
non-monotonic learning 105–18
NTC (New Term Constructor) program 294
numerical representation
  separation from logic-based representation 221
  transformation to logic-based representation 221, *222*
numeric attributes, domains with, trees induced from 198–9
NUT system 4, 12
  calculi of *13*

object models, KAUS's use of 323, *324*
object-oriented coordinate frame 232
  decision tree for 232, *233*
object-oriented programming, knowledge systems coupled by 6
object-pushing experiment
  coordinate frames used 230–2, 237
  generation of model in 232–6
  method used 228–30
Occam's Razor 232
oracle, expert acting as 130, 133, 135
ordered type, use by CHARADE 162–3
outliers, flagging in induction system 134
over-generalization, errors induced by 171–2

PANIC (Perception and Action as Naïve Causality) model *235*
  dynamic control using 239
  graphical representation of *236*
  knowledge hierarchy levels of *237*
partial match, learning situation 179, 180, 181
partial theory, benefits of constructing 170
partitionable, meaning of term 192
partitionworthy, meaning of term 192
passive observation, data acquired by 170
pattern matching
  chess positions 284
  protein structure problem 306
'person', knowledge representation for 11–12
pickup operator 209, 211, 212
PIR (protein) data base 307
place operator 209, 212
planners, robot control by 225–6
plausibility-based systems, weights of evidence in 188
plausibility coefficients 158
plausibility-gain trees; *see* evidence trees
plausibility trees
  backward pruning of *194*, 200
  conversion to/from evidence trees 192–4
plausible reasoning
  alternative approaches 56–7, 60–3
  argument-based model of 60–3
  calculi used 55
  complexity issues 59–63
  implementation examples 68–74
  knowledge representation issues 57–9
  probabilistic model of 55–6
  related approaches 63
PLS1 program 294
pointer stack 10
pole-balancing problem
  choice of means of control 248–50
  complexity reduced 246–8
  credit assignment algorithm used 226
  described 241–3
  qualitative model used 248–9, 251–2, 253–4, 257
    system parameters chosen 254–7, 258
  qualitative way of solving 241–58
  verification of solution 245–6
pole–cart system
  angles of posts controlled 249–50
  approximations used 247–8
  cart controlled whilst balancing poles 250–1
  choosing means of control 248–50
  control of 251–4
    cart while balancing poles 250–1
    chain of integrators 252, 253
    effect of delay 255, 256
    one integrator 252, 253
    total system 251–4
  control rule for 253–4
  described *242*
  division of quantity space for 253
  dominant path of control for *250*, 256, 257

drift behaviour modelled 251, *252*
effects of,
changing length or mass of poles 258
control parameters on system properties 256–7
forces acting on 244
hypothesis about operating region for 247
measurements required 242
modelling of 243–5
oscillation in 255–6
positive feedback loops in 249
qualitative model of controlled integrator 255–6
relevant properties of 243
semi-qualitative model of 248–9, 251–2
system parameters chosen 254–7
trial-and-error method used 255
Pop-2 memo function 196
predicate logic
knowledge representation using 318–19
set concept of 320–2
pressure regulator model
qualitative state of 263
rules for 265
single-level representation of 260, 262–5
structure of 262
prior knowledge, use in induction systems 124, 134, 135
priors, problems with 57
PRIZ system
applications of 44
architecture of 18–19
compared with PROLOG 18
direct specifier for 9
example problem solved by 21, 36
independent subtasks used 30–5
inheritance specifier for 10
input language used 19–21
knowledge base in 19
knowledge representation in 8, 9
logic united with procedural knowledge in 8–9
metalanguage used 9–10
preprocessor used 25–6
presentation of natural deductions from 26–7
program derivation rules for 35–6
program synthesis in 17, 18, 19, 23–5
structural synthesis rules for 31
use of program synthesizer as theorem prover 25–30
probabilistic calculus, language limitations in 58
probabilistic model (for plausible reasoning) 55–6
alternatives to 56–7, 60–3
complexity issues 59–60
knowledge representation issues 57–9
objections to 60
problem formulation task, interactive induction 133, 134
problem-solving capability, enhancement of 314–17
problem-solving process
example of *316*
formalization of 312–14
procedural knowledge, logic united with 7–9
procedural type atoms, use in KAUS 324–5
PRODIGY general-purpose planner 183–4
Program Debugging System 106–7
program optimization, inductive synthesis applied to 147
program synthesis, PRIZ's use of 17, 18, 19, 23–5
PROLOG
cardiac diagnostic algorithm written in 269–70
Closed World Specialization Algorithm implemented in 112–13
compared with PRIZ 18
computational models in 39–47
debugging of programs 171, 172–3
'deriv' function implemented in 117–18
Edinburgh syntax conventions of 214
extension of 15, 43
interpreters 68
SLDC trees found by 67
logic united with procedural knowledge in 7–8
meta-interpreter used 43–4
negation-by-failure used 108, 111
robot programming using 206, 227
StructureMover program implemented in 213–15
PROMIS (protein machine induction system) program 291, 292
alpha-helix prediction rule 301–3
chemical knowledge coded into 301
experimental results 299–307
data used 299
general rules for predicting secondary structure 299–303
threshold logic used 305–7
variation of successful rules 303–5
input to 295
learning by 296
operators used 297
output of 295
rule evaluation by 297–8
search control in 298–9
threshold test used 298
prompting, interactive induction 127

propositional Horn clause logic 17
  program using, characteristic sample set generated for 99, 100
  see also PRIZ
propositional logic programming 17–36
propositional variables, PRIZ's use of 10, 21–2
proposition surrogates
  concept first introduced 85, 88
  example of use 86–7
  first member of 86, 87
PROSPECTOR expert system 55, 59
  Bayesian inference used in 187–8
  Boolean combinations of events treated by 70
  combination of multiple evidence treated in 70
  if–then rules used 69–70
  plausible reasoning mechanism used by 69–71
  probabilistic inference mechanism of 69–71
    implementation of 70–1
protein folding
  learning applied to prediction of 291–308
  problem to be solved 292–3
proteins
  alpha-helix secondary structure prediction 301–3
  data bases of structure information 293, 307
  primary structure of 292
  secondary structure of 292
  secondary structure prediction 292, 293, 294–308
    chemical theory methods used 294
    general rules found 299–303
    homology-based methods used 294
    statistical methods used 294
    threshold logic used 305–7
    variation of successful rules 303–5
  types of 292
protein structure problem
  alpha-helix prediction rule 301–3
  mistake matching used 306
  previous work done 294
  suitability of problem 293
proximity scanning technique 209, *210*
proximityscan operator 209, 212
P-space-complete problems 8, 17, 30
Puma robots 206, *207*, 227, *228*

QSIM qualitative simulator 241
qualitative modelling, varying levels of abstraction in 259–79
qualitative models
  experimental results using 273–6
  interpretation of 268–72
  representation as hierarchy of models 265–8
  representation of 261–8
  research on machine-aided construction of 279
  single-level 262–5
qualitative representations (of knowledge) 203–79
quantity space, division for pole-balancing problem 253
Quinlan method, decision trees built-up by 157

reactive environment
  learning in 170–84
  meaning of term 169, 179
recursive programs, construction in PRIZ 17
refinement operator 175
refutation trees, transformation into derivation trees 114–15
regularities, detection by CHARADE 157–9, 165
resolution
  clause entailment using 110, 114
  unification within theory 97
resolution closure, definition of 109
resolution principle, inversion of 93–102
resolution theorem, Robinson's results on, 109
Rhino robot *207*
right-to-left goal selection strategy 268, 271
ROBCON program 206, *207*, 227, 229
ROBEYE program 206, *207*, 227, 229
robotics, definition of 205
robot learning, CAP used 179–83
robot programs, two-phase approach to building 227
robots
  learning of causality by 225–41
    coordinate frame 230–2, 237
    experimental method 227–30
    model generation 232–6
  response to change 238
  structure moving by 208–20
  user-interaction facility for 221–3
ROPRO system *226*
rule evaluation, protein structure problem 297–8
RULEGEN, threshold test in 298
rule induction, table compression using 198

RuleMaster inductive learning system 206, *207*
rule presentation task, interactive induction 133–4
rule properties, learning bias affected by 159–60
rules
  expert's modification of 129–30, 134
  flexible interpretation of 306
  protein structure problem 295
rule system
  a priori structure of 160
  learning with 153, 156–61
run-time
  compatibilities between belief-states and truth-values encountered at 195
  learning occurring at 195–6
Russell's paradox 319

S4 modal logic 17, 30–5
  derivability in 32–4
safegrasp procedure 216–17
sample computations
  direct screen display of 146–7
  general algorithms restored from 140, 142
  WHILE-expressions constructed from 144
scientific discovery, application of machine learning to 291–2
search control, PROMIS's use of 298–9
search mechanism, DUCE's use of 101–2
search operators, background knowledge on proteins used as 296–7
semantic networks 5–6
  inference on 6
  time-relations in 5
'sem' function, definition of 11, 22–3
semi-decision trees 190, 196
  conversion from plausibility trees 194
sense and denotation, formal language based on 81
sense of name, meaning of term, 80n
sensory information, robot control using 207, 226
sequential Bayes, used with class probability trees 187–200
SEQUOIA, threshold test in 298
Shapiro–Kopec expert system 94, 95
Shapiro's debugging algorithm 172–3, 174
short-term memory, capacity of 284
signal flow graphs 315, *316*, 328
silhouette identification, C4 compared with Evidencer algorithms 197, 198–9
single-concept learning systems 169
skolemization 114, 117
SLDC tree
  definition of 66
  index of 66–7
  PROLOG interpreter to find 67
  see also certainty calculus
software facilities, induction tools 135
specialization
  closed world 111–13
  meaning of term 106
  operators required 175
  protein structure problem 297
specification languages; *see also* UTOPIST...
specifications, axiomatic semantics of 21–3
'square' (computational) model 39
  calculating programs called from 43
  encoding of 41
  implementation in PROLOG 43–4
  use of 42–3
  writing of 40
statistical methods, protein structure predicted using 294
stratification of knowledge *14*
STRIPS planner 225–6
Structural Synthesis Rules (in PRIZ) 23, 35
  with independent subtasks 31
structured induction technique 188
StructureMover program 207
  algorithms in 209–17
  combinatorial complexity problems 215, 219
  commongrasp procedure 216
  constraints with respect to structures 219
  extensions possible 220–1
  interpret procedure 215–16
  limitations due to inaccuracy in measurements 220
  limitations with respect to grippers 220
  performance and limitations of 217–21
  PROLOG implementation of 213–15
  safegrasp procedure 216–17
structure-moving task 208–9
  algorithms used 209–17
  lookinto operator for 209, 212–13
  pickup operator for 209, 211, 212
  place operator for 209, 212
  proximityscan operator for 209, 212
structures, exploring of 205–23
subcommittees model 195–6
subjective information
  meaning of term 122
  methods of providing 122, 124
subsumption theorem 113
Subtasks, dependent compared with independent 30
suicide example 118
superstable structure, meaning of term 215

supervision, expert system performance affected by 95
surface models, compared with deep models 264–5
symbol reduction, DUCE operators giving 94

'tantrum' (by learning system) 183
term rewriting system 51
theorem prover
  PRIZ's program synthesizer used as 25–30
  examples 27–30
theory
  learning systems,
    correcting errors in 174–7
    definition of 170
    locating errors in 172–4
    maintaining consistency in 170, 177–9
    recognizing existence of errors 172
  as network of concepts 172, *173*
theta-subsumption 110
threshold logic, use in protein structure problem 305
threshold test 298
translation (of formulas and sequents) 31
travelling expenses claims
  certainty lattice used 72
  home/abroad distinction 72, 73, 74
  user interface strategy 74
truth maintenance systems 14, 58, 63
truth values, compatibility with belief states at run-time 195
typing, attributes affected by 164

uncertain inference, PROSPECTOR's support of 189
unification 97
unifying terms, construction of 49–53
user interfaces
  expert system 74
  KAUS 326, *327*
  knowledge-base bootstrapping 126
UTOPIST (Universal Translator Of Problems Including Specifying Texts)
  input language 18, 19
  example specifications in 19–21
  semantics of 21–2

validation, induced knowledge 132–3, 134
VAL II robot programming language 206
Variable Precision Logic 273
Version Space algorithm 106
voting patterns, C4 compared with Evidencer algorithms 197, 199

Wald sequential cut-off method 187, 200
Wang's algorithm 72
WARPLAN 220
weight of evidence
  updating using 70
  see also plausibility gain
WHILE-expressions, inductive synthesis 143–4
window-oriented coordinate frame 232
  decision tree for 232, *233*
world model
  errors in,
    correcting errors 174–7
    locating errors 172–4
    recognizing existence of errors 172
  evolution over time 170
  maintaining consistency in 170, 177–9

XpertRule, induction of evidence trees implemented by 196–7

YAPES expert system shell 71, 74, 75
  user interface strategy 74

Zermelo–Fraenkel set theory 319